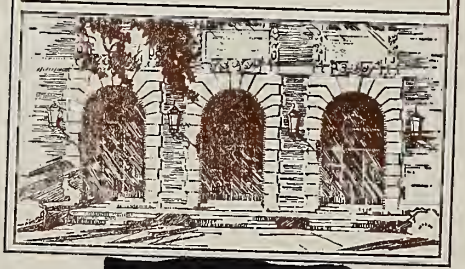
LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
Il6r
no. 715-721
cop. 2



Digitized by the Internet Archive
in 2013

http://archive.org/details/multiprocessorne717iway

510.84
I16r
no. 717

Math

9

UIUCDCS-R-75-717

MULTIPROCESSOR NETWORK SYSTEM DESIGN BY PROCESS CONCEPT

by

Akihiro Iwaya

Dup.

December, 1975

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-75-717

MULTIPROCESSOR NETWORK SYSTEM DESIGN BY PROCESS CONCEPT

by

Akihiro Iwaya

December, 1975

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

This work was supported by the Department of Computer Science and was submitted in partial fulfillment for the Master of Science degree in Computer Science, 1975.

ACKNOWLEDGMENT

The author would like to express his gratitude to his advisor, Professor Jane Liu, for her guidance during the preparation of this thesis and also her careful reading and valuable suggestions for the improvement of the original manuscript. In addition the author owes a debt of gratitude to his wife, Masayo, whose understanding made this thesis possible in a limited period of time.

This paper is dedicated to his two children, Miki and Masaaki.

TABLE OF CONTENTS

	Page
1. INTRODUCTION	1
2. AIRLINE RESERVATION SYSTEM	3
2.1 System Requirements	4
2.2 Terminal Operation	4
2.3 Central Computer Operations	8
2.3.2 File Structure	11
2.4 Summary	13
3. FILE SYSTEM DESIGN	15
3.1 File System Organization	15
3.2 Logical File System (LFS)	16
3.3 Operation of File System	20
3.4 External Search Routine	24
3.5 Command Handling Routine	31
3.6 Summary	32
4. CENTRAL COMPUTER DESIGN	38
4.1 Summary of Operation	38
4.2 System Design	45
4.2.1 System Decomposition Rules	45
4.2.2 Decomposition of Reservation System into Processes	45

Page

4.2.3 Operation Flow by Process Cooperation..... 48

4.3 Reservation System Processes..... 49

4.3.1 Line Input Process..... 49

4.3.2 Interactive Job Handler Process..... 51

4.3.3 Line Output Process..... 54

4.3.4 Scheduler Process..... 54

4.3.5 Command Handler Process..... 57

4.3.6 Edit Process..... 59

4.3.7 File Management Process..... 61

4.3.8 External Search Process..... 64

4.3.9 Probe-delete Process..... 67

4.3.10 File I/O Process..... 70

4.4 Hardware Architecture and Operating System..... 70

4.4.1 The Rules of Organizing Cluster Network..... 70

4.4.2 Cluster Network Organization..... 70

4.4.3 Local Operating System..... 76

4.4.4 Node Processor Organization..... 77

4.4.5 Implementation of Kernel Routines..... 78

4.5 Summary..... 80

5. CONCLUSION..... 81

LIST OF REFERENCES..... 83

1. INTRODUCTION

During recent years, a great deal of efforts have been made in the development of computer network systems leading to the design and implementation of several centralized computer network systems [1,2,3]. Typically, such a network contains several miniprocessors (or microprocessors) connected together to form a computing system whose performance is equivalent to a large scale computer system while offering such advantages as increase of reliability, expandability and cost reduction.

This report describes the design of a dedicated multiprocessor cluster network [4] which functions as an airline reservation system. The design approach used is as follows: we decompose the system into cooperative processes. A dedicated processor is assigned to each process and processors are connected via suitable inter-process communication links to form a cluster network. The airline reservation system is selected to demonstrate these design approaches since the functions performed in the reservation system are fixed. Hence the problem of process decomposition is relatively straightforward. Moreover, it being an interactive system, we need to be concerned with many interesting aspects in operating system design, such as protection mechanisms, file structures, etc.

In Chapter 2, a hypothetical airline reservation system is described. The terminal operations in the system and the central computer

operations in an interactive job mode are also discussed. Chapter 3 describes the file system, which is the most crucial part of such an interactive system. The design of the central computer based on process cooperation concept is presented in Chapter 4. The structure of the cluster network and the required characteristics of the system are also described.

2. AIRLINE RESERVATION SYSTEM

In an airline reservation system, customers' requests are sent through communication lines from the terminals at sales offices to the central computer in the central reservation office. The sales offices are usually scattered over a wide area geographically. Hence, messages from these sales offices are often gathered in a message switching center before being sent to the central office. A typical airline reservation system is organized as a tree shaped network with the root representing the central office. Such systems are now used by most of the major airline companies.

Many functions required in an airline reservation system are executed interactively between the computer and the operators at sales offices in a foreground mode. For example, an inquiry asking available seats on a given flight must be answered in 2 to 3 seconds. The airline reservation system must offer the capability of meeting this response time requirement. We believe that such a design goal can be achieved cost-effectively in a multiprocessor cluster network. To determine the hardware organization of a cluster network and the requirements of an operating system, a simplified model of an airline reservation system is considered. The system requirement is presented in section 2.1. The terminal operations and the central computer operations are discussed in sections 2.2 and 2.3, respectively.

2.1 System Requirements

Throughout this paper, we consider a system designed to suit the need of a medium-sized airline company. The relevant characteristics of this company are as follows:

1. The central office has a central computer with a large file containing information on all schedule flights.
2. There are 40 offices with teletype-like terminals.
3. There are 1000 flights per day; each flight has a capacity of 200.
4. Reservations can be made 1 year in advance.

2.2 Terminal Operation

References to the data base are made whenever a customer arrives at or calls the sales office. The operator keys in and sends a terminal command which refers the data base, and in reply, he receives a response from the central computer. The terminal command types which can be sent from a sales office are as follows:

1. Start operation (CM1): conversation starts (job control command);
   Terminal input: Command;
   Response: Acknowledgment;
2. Available seats inquiry (CM2): ask for information about the available seat;
   Terminal: Command, Flight number, Flight date;
   Response: Acknowledgment with the number of seats available;

3. New reservation (CM3): reserve the seats of particular flight;

   Terminal input: Command, Flight number, Flight date, Number of seats, Passenger's name, Address, Phone number;

   Response: Acknowledgment with the number of seats reserved;

4. Cancellation: cancel the reservation;

   Terminal input: Command, Flight number, Flight date, Number of seats, Passenger's name;

   Response: Acknowledgment with the number of seats remaining;

5. Confirmation (CM5): confirm the reservation and issue the ticket;

   Terminal input: Command, Flight number, Flight date, Passenger's name;

   Response: Acknowledgment with the number of seats reserved;

6. Passenger's information inquiry (CM6): request the passenger's information;

   Terminal input: Command, Subcommand* (Address, Phone number), Flight number, Flight date, Passenger's name;

   Response: (Address, Phone number), Reservation state;

---

*This shows what kind of information items the operator wants.

7. Flight information inquiry (CM7): Request the information by origin and destination;

   Terminal input: Command, Subcommand* (Time, Fare, Operation date, Meal, Equipment), Origin, Destination;

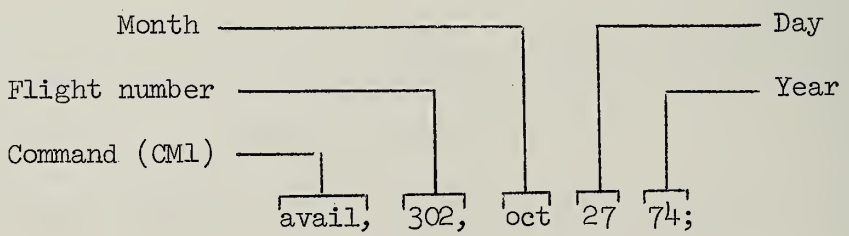
   Response: Flight number (Departure time, Arrival time, Fare, Operation date, Equipment);

8. Stop operation (CM8): Conversation ends (Job control command);

   Terminal input: Command;

   Response: Acknowledgment;

Normally one transaction consists of few terminal commands including CM1 (start) command and the CM8 (stop) command. A typical terminal command is structured in the following way:



All terminal commands are typed in lower case letters. Response from the central computer are typed in upper case letters, as in conventional conversation systems.

*This shows what kind of information items the operator wants.

Examples of terminal command format of different command types are shown as follows:

CM1: Input start;

Response START OK

CM2: Input avail, 302, oct 27 74;

Response AVAIL NUM 020

CM3: Input reserve, 302, oct 27 74, 004, akihiro iwaya, 2023c orchard-st. urbana ill. 217-384-1919;

Response RESERVED, SEAT 004

CM4: Input cancel, 302, oct 27 74, 004, akihiro iwaya;

Response CANCELED, SEAT 000

CM5: Input firm, 302, oct 27 74, akihiro iwaya;

Response CONFIRMED, SEAT 004

CM6: Input passenger address, 302, oct 27 74, akihiro iwaya;

Response 2023c ORCHARD-ST. URBANA ILL., RESERVED

CM7: Input flight time, chi, saf;

Response 265 09:00A 11:25A 291 01:00P 04:51P 213 01:20P 03:43P 215 04:55P 07:22P 047 07:15P 09:39P

CM8: Input stop;

Response STOP OK

An example of typical transactions is shown below:

Transaction example:

```
start;                                                    Type in CM1
START OK                                                  Response
flight time, chi, saf;                                    Type in CM6
265 09:00A 11:25A 291 01:00P 04:51P                       Response
213 01:20P 03:43P 215 04:55P 07:22P
047 07:15P 09:39P
avail, 215, nov 30 74, 004;                               Type in CM2
AVAIL NUM 010                                             Response
cancel, 302, oct 27 74, 010, richard nixon;               Type in CM4
CANCELED, SEAT 000                                        Response
```

```
firm, 302, oct 28 74, gerald ford;                    Type in CM5
CONFIRMED, SEAT 005                                   Response
reserve, 215, nov 30 74, 004, akihiro iwaya;          Type in CM2
2023c orchard-st. urbana ill. 61801, 217-384-1919
RESERVED, SEAT 004                                    Response
firm, 215, nov 30 74, 004, masayo iwaya;              Type in CM5
PASSENGER NOT IN FILE                                 Response
stop;                                                 Type in CM7
STOP OK                                               Response
```

This example shows a transaction consisting of a chain of terminal commands. The first command was an inquiry about flight information and a question on available seats. While the customer was probably thinking about making a reservation, other customers' cancellations and confirmations were made. After that, the reservation for the first customer was done. Then the confirmation was made with a wrong name and the computer responded with the error message.

2.3 Central Computer Operations

Transactions from sales offices are sent to the central computer in the computer center through data transmission lines. During each transaction, the central file is accessed. In order to design the central computer operation in detail, its requirements must be made explicit. In this section, the file access operations required to process different transactions are described. The other operations which are also required for the reservation system are described in Chapter 4.

2.3.1 Operations Required to Process the Terminal Command

The following file accesses operations occur when different terminal commands are processed by the central computer:

CM2 Available seats inquiry:

1. Search for the count of available seats with flight number and flight date as a searching key.

2. If there is no such key, return the code which shows that the key is not in file and then return.
3. Return the count of available seats.

CM3 New reservation:

1. Search for the count of available seats with flight number and flight date as a searching key.
2. If there is no such key, return the code which shows that the key is not in file and then return.
3. Compare the count of available seats with the number of seats required.
4. If the number of seats required is greater, return the code stating that there are no seats available and then return.
5. If the number of available seats is greater, search for the entry where passenger's record is stored.
6. Decrease the count of available seats.
7. If passenger's name is already in file, increase the number of reserved seats. Otherwise, put passenger's name, address and phone number with the number of new reserved seats.
8. Return the code stating that the seats are successfully reserved with the number of reserved seats.

CM4 Cancellation:

1. Search for the passenger's record with flight number, flight date and passenger's name as a key.
2. If there is no such key, return the code which shows that the key is not in file and then return.

3. Otherwise, decrease the number of reserved seats in the passenger's record.
4. If the number of reserved seats is negative return the error code and then return.
5. Increase the count of available seats.
6. If the number of reserved seats is zero, remove the passenger's record.
7. Return the code which shows that the reservation is canceled with the number of remaining seats.*

CM5 Confirmation:

1. Search for the passenger's record with flight number, flight date and passenger's name as a key.
2. If there is no such key, return the code which shows that the key is not in file and then return.
3. Return the code stating the number of seats reserved.

CM6 Passenger information inquiry:

1. Search for the passenger's record with flight number, flight date and passenger's name as a key.
2. If there is no such key, return the code which shows that the key is not in file and then return.
3. Get the items of information which match the desired data from the passenger's information record specified by subcommand.
4. Return the requested items of information.

---

*If the number of remaining seats shows zero, it means total cancellation was made.

CM7: Flight information inquiry:

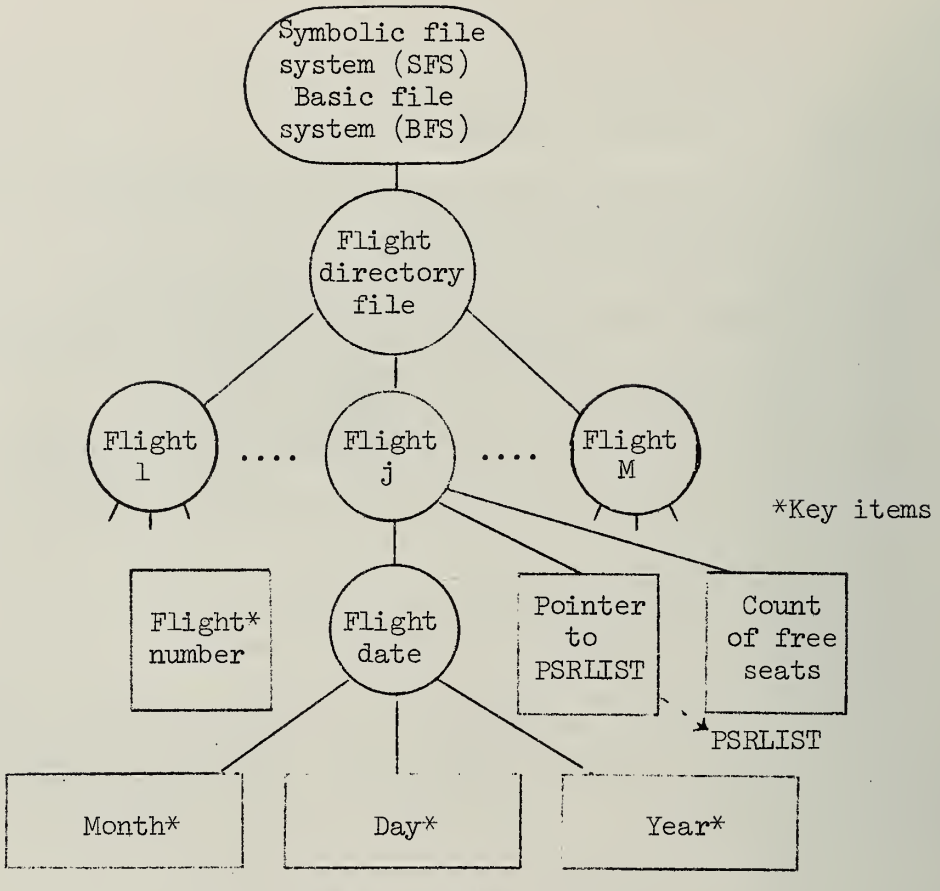
1. Search for the flight information record with origin and destination as a key.
2. If there is no such key, return the code which shows that the key is not in file and then return.
3. Get the items of information which match the desired data from the flight information record specified by subcommand.
4. Return the requested items of information.

CM1 (start) and CM7 (stop) are job control commands. These commands are used to open and close the transaction. These commands will be discussed in Chapter 4.

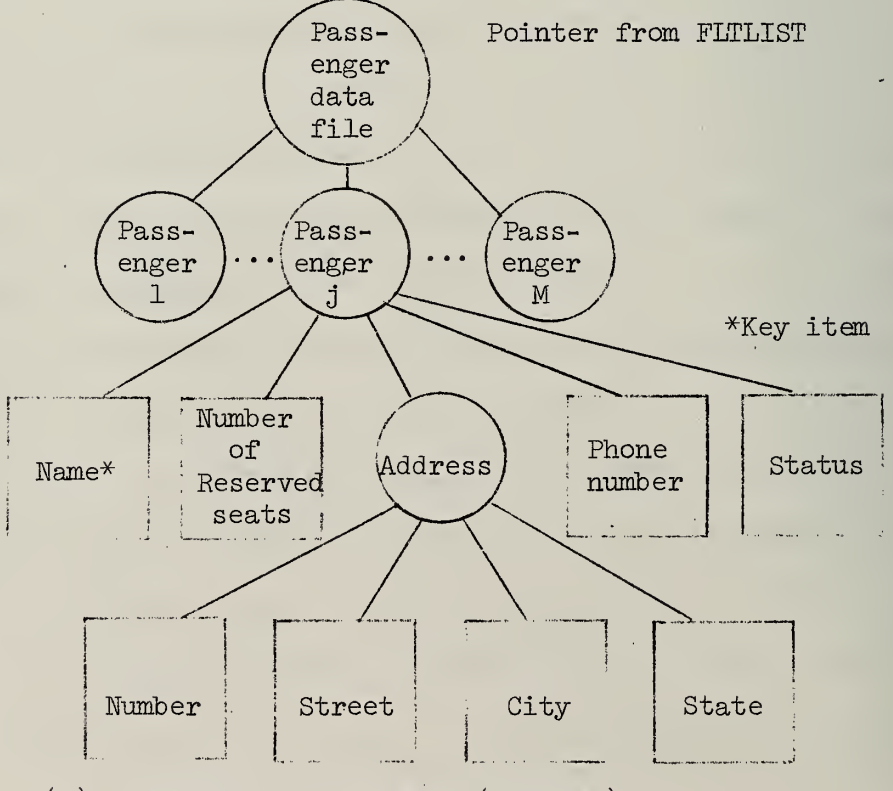
2.3.2 File Structure

We present here an overview of file structure.

Two files, passenger information file and flight information file, are required in this system. The passenger information file stores the personal information of all passengers who reserved the seats for each flight. This file is further divided into two subfiles, flight directory file and passenger data file. The flight directory file contains for each flight the count of free seats and a pointer to the location of the passenger data file. The passenger data file stores information on each passenger who holds reservations on the particular flight. Fig. 2.1 shows the file structure of the flight directory file (FLTLIST) and the passenger data file (PSRLIST). Two level searching must be performed to obtain a passenger's records. First level search is performed



(a) Flight Directory File (FLTLIST)



(b) Passenger's Data File (PSRLIST)

Figure 2.1 Structure of Passenger's Information File

using two items, flight number and flight date, as search keys in the FLTLIST. As a result the pointer for the location of the PSRLIST of the particular flight is obtained. The second level search is performed using passenger's name as a search key in the PSRLIST and other personal information of the particular passenger are obtained. Note that the count of free seats is stored in the FLTLIST; hence this item is obtained in the first level search.

The flight information file is used to store all information of scheduled flights. Fig. 2.2 shows the file structure of the flight information file (INFOLIST). Origin (Departure.place) and destination (Arrival.place) are used as search keys to obtain other flight information items.

Note that in Fig. 2.1 and Fig. 2.2, only terminal nodes of trees represent actual data stored in the specified locations of file storage.

2.4 Summary

A simplified model of an airline reservation system was developed in this chapter. The terminal operations were defined in terms of the specifications of terminal command. The central computer operations and the file structures which are used to handle these terminal commands were presented.

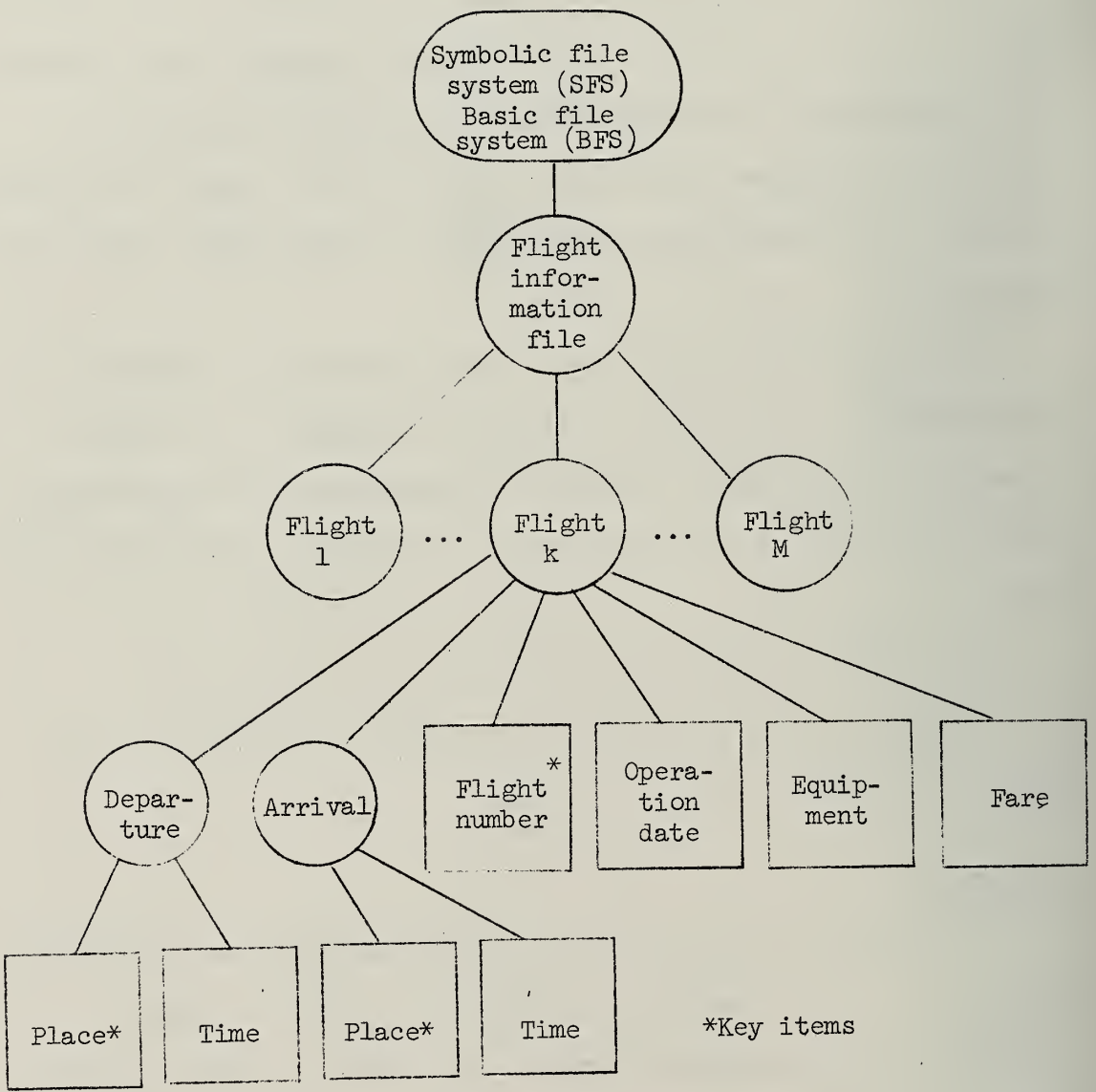


Figure 2.2 Structure of Flight Information File (INFOLIST)

3. FILE SYSTEM DESIGN

Passenger information and scheduled flight information are stored in a secondary memory. This information is read and updated corresponding to the requirement of terminal commands. The file system which manages the access to this information contains a hierarchy of modules [5]. These modules are: Symbolic File System (SFS), Basic File System (BFS), Logical File System (LFS), Physical File System (PFS), and File Device Management (FDM). In Section 3.1, the file system organization is presented. The detailed structure of the logical file system (LFS) is discussed in Section 3.2. The file system operation and search operation designed are presented in Section 3.3. Finally the complete operation for handling terminal commands based on the results of file system design is presented in Section 3.4.

3.1 File System Organization

We describe here the modules in the file system:

1. Symbolic File System (SFS), Basic File System (BFS)

   These two modules offer a mechanism to convert a symbolic file name to a file location. In the reservation system, all files for interactive operations are assumed to be always open. The open file names are maintained in a main-memory area called an Active File Directory (AFD) with the associated file locations. Hence, the locations

of files which need to be accessed in terminal transactions can be obtained from the AFD in main-memory area.

2. Logical File System (LFS)

   This module offers the database structure within a file. Detailed design of the LFS and the search method are presented later in this chapter.

3. Physical File System (PFS)

   This module offers the following functions:

   (i) Conversion of a logical file location to a physical file location.

   (ii) Management of file I/O buffer.

   (iii) Blocking and deblocking.*

4. File I/O Management

   This module contains a file I/O dispatcher which monitors I/O routes and device status and sends file I/O command to devices. It also contains a file I/O scheduler which chooses one file I/O request to run.

The PFS module and the file I/O management module are both implemented in a file I/O process presented in Chapter 4.

3.2 Logical File System (LFS)

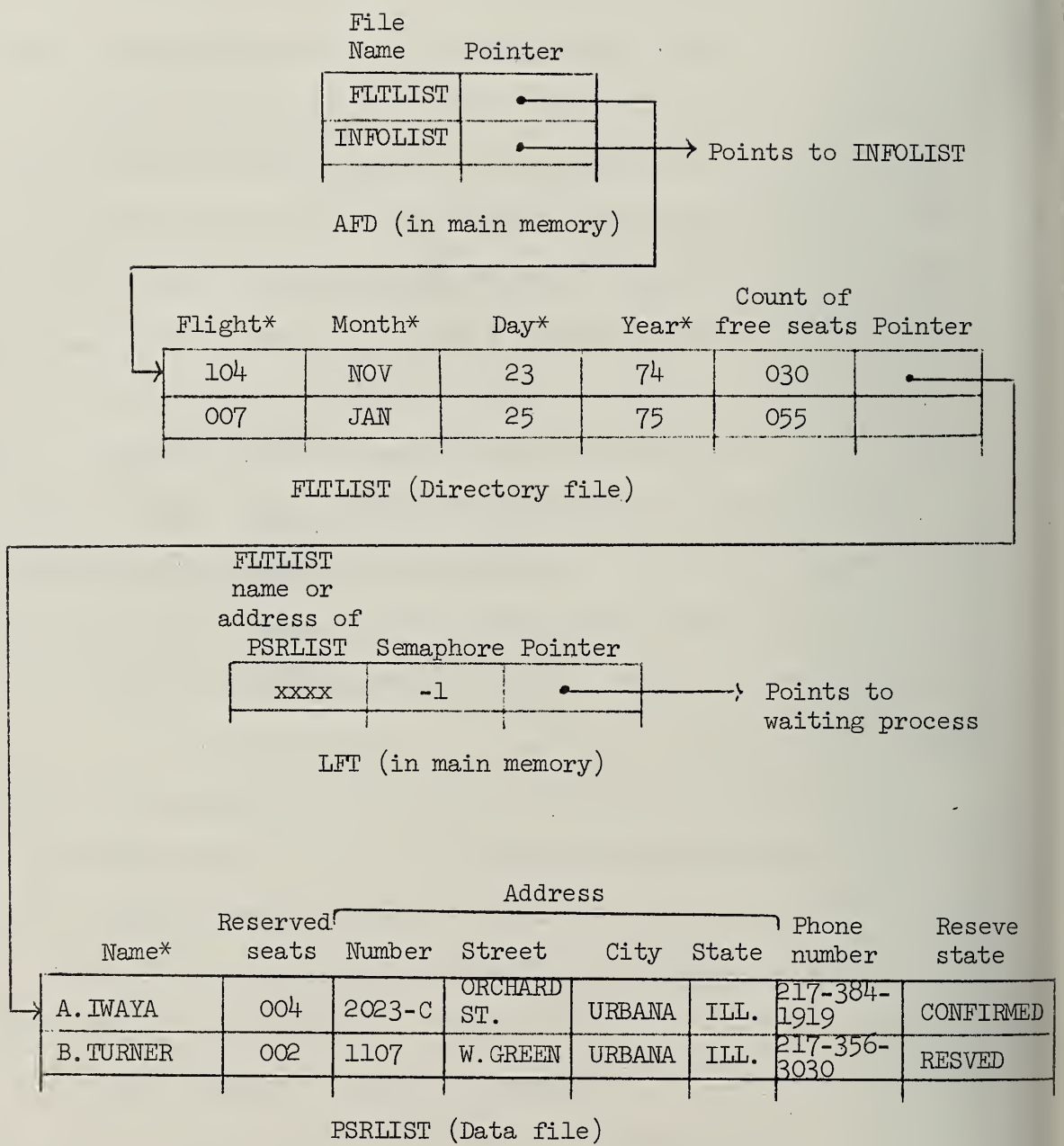
Logical file structure described here was designed so that the number of file I/O operations and the required file memory space are as

---

*File memory space is divided into pieces of the same size, called blocks.

small as possible. Record types were selected so as to use the file memory space efficiently. The search method was chosen so that search speed is fast enough to satisfy the requirement of response time. Finally, a protection method was designed such that only those file areas which need protection are protected while a process is using the file. To satisfy these requirements, a logical file is further divided into subfiles. These subfiles are organized into a hierarchy of subfiles. A higher level file is used as a directory file to refer to a lower level file. The lowest level subfiles are called data files. A subfile itself is organized into keyed records. All records are of fixed length. Hence, a length must be selected such that the maximum number of record-items could be stored. An open scatter hash address search algorithm [6] together with an external bucket is used. Lock mechanism selected is a method by a semaphore lock-operation on each file.

The logical file system for the reservation system consists of two major files: a passenger information file and a flight information file. Fig. 3.1 and Fig. 3.2 show the overall file structures designed for this system along with example record items. It consists of five data structures: active file directory (AFD), lock file table (LFT), flight directory file (FLTLIST), passenger data file (PSRLIST), flight information file (INFOLIST). FLTLIST and PSRLIST are the subfiles of the passenger information file. LFT is used to lock the file while a process is using the file. These structures are expressed in PL/I like declarations in the following way:



*Items used as a key

Figure 3.1 File System Data Structure (1)
(Passenger Information File)

	Dep. place	Arriv. place*	Flight number	Dep. time	Arriv. time	OP. date	Equip.	Fare
→	CHI	SAF	265	09:00A	11:25A	DAILY	B-727	$126.00
	CHI	SAF	291	01:00P	04:51P	DAILY	B-727	$126.00
	CHI	SAF	213	01:20P	03:43P	M,W,F	B-727	$126.00

*Items used as a key

Figure 3.2 File System Data Structure (2)
(Flight Information File)

```
1 AFD(*) CTL                                    /* ACTIVE FILE DIRECTORY   */
  2 FILENAME    CHAR(10),
  2 CHAIN       PTR;                            /* POINTS OPENED FILE      */
1 LFT(*) CTL                                    /* LOCKED FILE TABLE       */
  2 LFA         FIXED BIN(32),                  /* LOCKED FILE ADDRESS     */
  2 SEMAPHORE   FIXED BIN(8),                   /* SEMAPHORE FOR FILE LOCK */
  2 CHAIN       PTR;
1 FLTLIST BASED,                                /* FLIGHT DIRECTORY FILE   */
  2 FLIGHT(*)   CTL,
    3 NUM       CHAR(3),                        /* FLIGHT NUMBER           */
    3 DATE,                                     /* FLIGHT DATE             */
      4 MONTH   CHAR(3),
      4 DAY     CHAR(3),
      4 YEAR    CHAR(2),
    3 FRSEAT    CHAR(3),                        /* COUNT OF FREE SEATS     */
    3 CHAIN     PTR;                            /* POINTER TO PSRLIST      */
1 PSRLIST BASED,                                /* PASSENGER DATA FILE     */
  2 PSR (200),                                  /* PASSENGER               */
    3 NAME      CHAR(20),
    3 RVSEAT    CHAR(3),                        /* RESERVED SEAT           */
    3 ADDRESS,
      4 NUM     CHAR(6),
      4 STREET  CHAR(15),
      4 CITY    CHAR(15),
      4 STATE   CHAR(4),
    3 RVSTATE   CHAR(4);                        /* RESERVED/CONFIRMED      */
1 INFOLIST BASED,                               /* FLIGHT INFORMATION FILE */
  2 FLIGHT (1000),
    3 ORIGIN;
      4 PLACE   CHAR(3),
      4 TIME    CHAR(5),
    3 DESTIN,
      4 PLACE   CHAR(3),
      4 TIME    CHAR(5),
    3 NUM       CHAR(3),                        /* FLIGHT NUMBER           */
    3 OPDATE    CHAR(5),                        /* OPERATION DATE          */
    3 EQUIP     CHAR(5),                        /* FX 'B-727'              */
    3 FARE      CHAR(6);                        /* IN $                    */
```

3.3 Operation of File System

The file operations for the file structure presented in Section 3.2 are: read, multi-read,* rewrite, and delte. Each operation is accompanied

*All records with the same key are read.

by a record search operation with keys. The statement which calls the file operation may look like the following:

READ (<File name>) INTO (<Variable>) KEY (<Key name>) Lock;

The operations performed to obtain a desired record in a file system are described in Figs. 3.3 and 3.4.

1. The file name is searched for in the AFD to obtain the address of the file.
2. If the name is not found, the SFS routine and the BFS routine are further called.
3. If the address of the file is given to the file system, step 2 and 3 are skipped.
4. If this file is a type of file which needs a lock operation, the LFT checks whether this file is already locked. If this file is locked, the operation is kept waiting until it is unlocked. If this file is not locked, the lock operation is performed and proceeds to step 6.
5. However, if this file is not a type of file which needs a lock operation, step 4 is skipped.
6. External search operation is performed.
7. If specified operation is a read or multi-read operation, the record searched for is moved to working area.
8. If the specified operation is a new-write or rewrite operation, the record in working area is moved to file output buffer and then written into the area in a secondary memory.

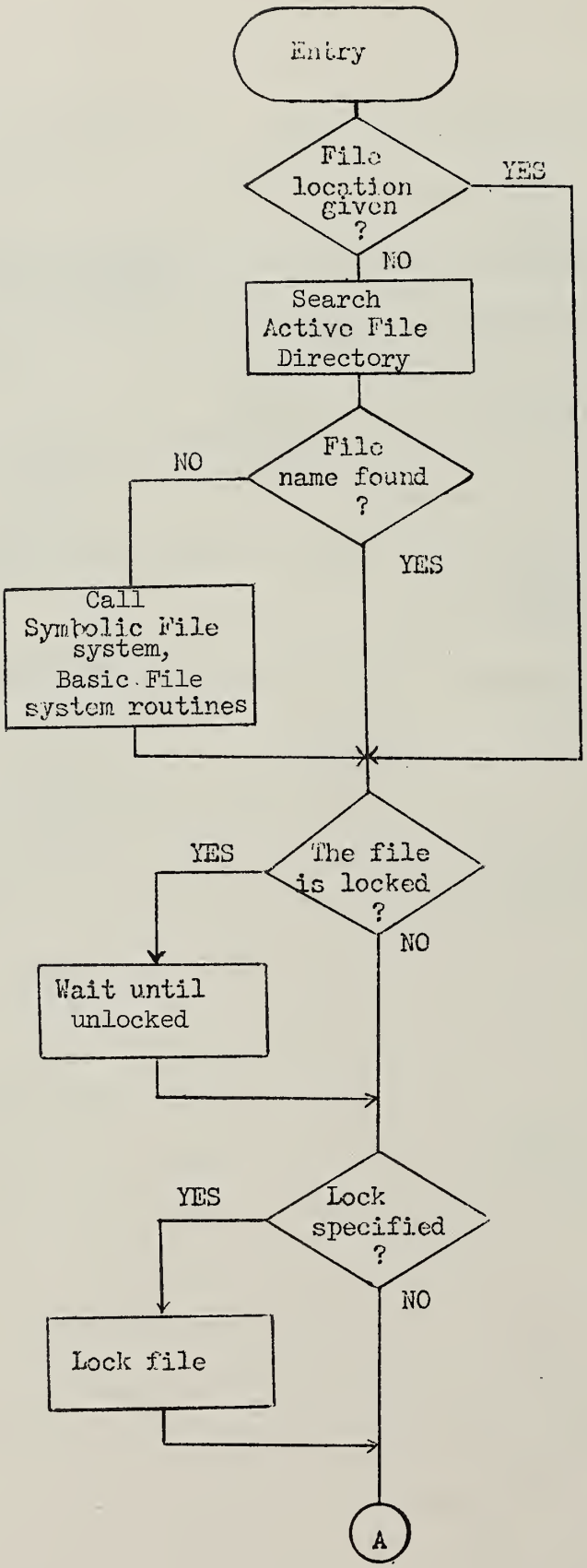


Figure 3.3 The Flowchart of File Management Routine (1)

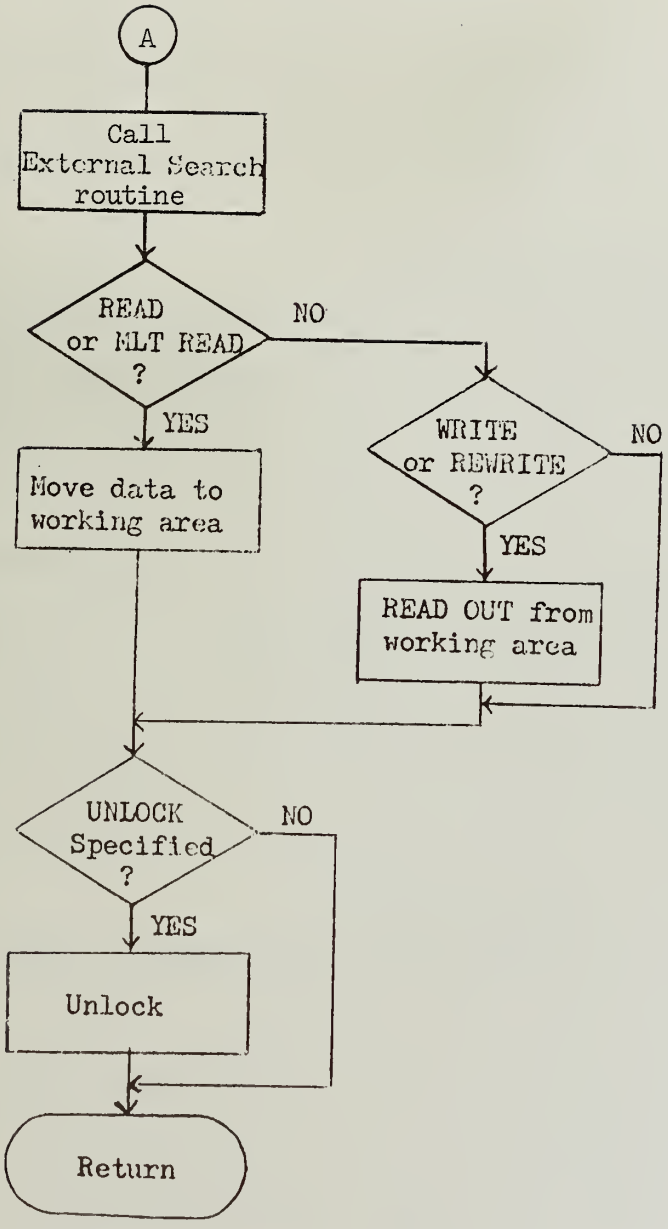


Figure 3.4 The Flowchart of File Management Routine (2)

9. If the specified operation is a delete operation, step 7 and 8 are skipped.
10. If the unlock is specified then the unlock operation is performed.

3.4 External Search Routine

The external search routine designed for this system is a modified version of the open scatter hash addressing search algorithm. The environment in which this routine is executed is schematically shown in Fig. 3.5.

Let 'M' be the total number of buckets, 'b' be the number of entries in a bucket and assume each entry consists of 'n' bytes. In order to access a particular record in a table (logical file), a hash value 'i' is calculated from the given key, then the logical file address (LFA) of the bucket is:

$$LFA = i \times b \times n$$

The physical block address (PBA) which includes this bucket is:

$$PBA = FHA + \lfloor LFA/k \rfloor$$

Where FHA is the file head block address which can be obtained from the AFD (active file directory) and k is the number of bytes in a physical block. The bucket address (BA) within a physical block is:

$$BA = LFA \bmod k$$

The PBA (physical block address) is used as a secondary memory location to move the physical block in a secondary memory to the buffer area and the BA (bucket address) is used to move a bucket from the buffer area to the

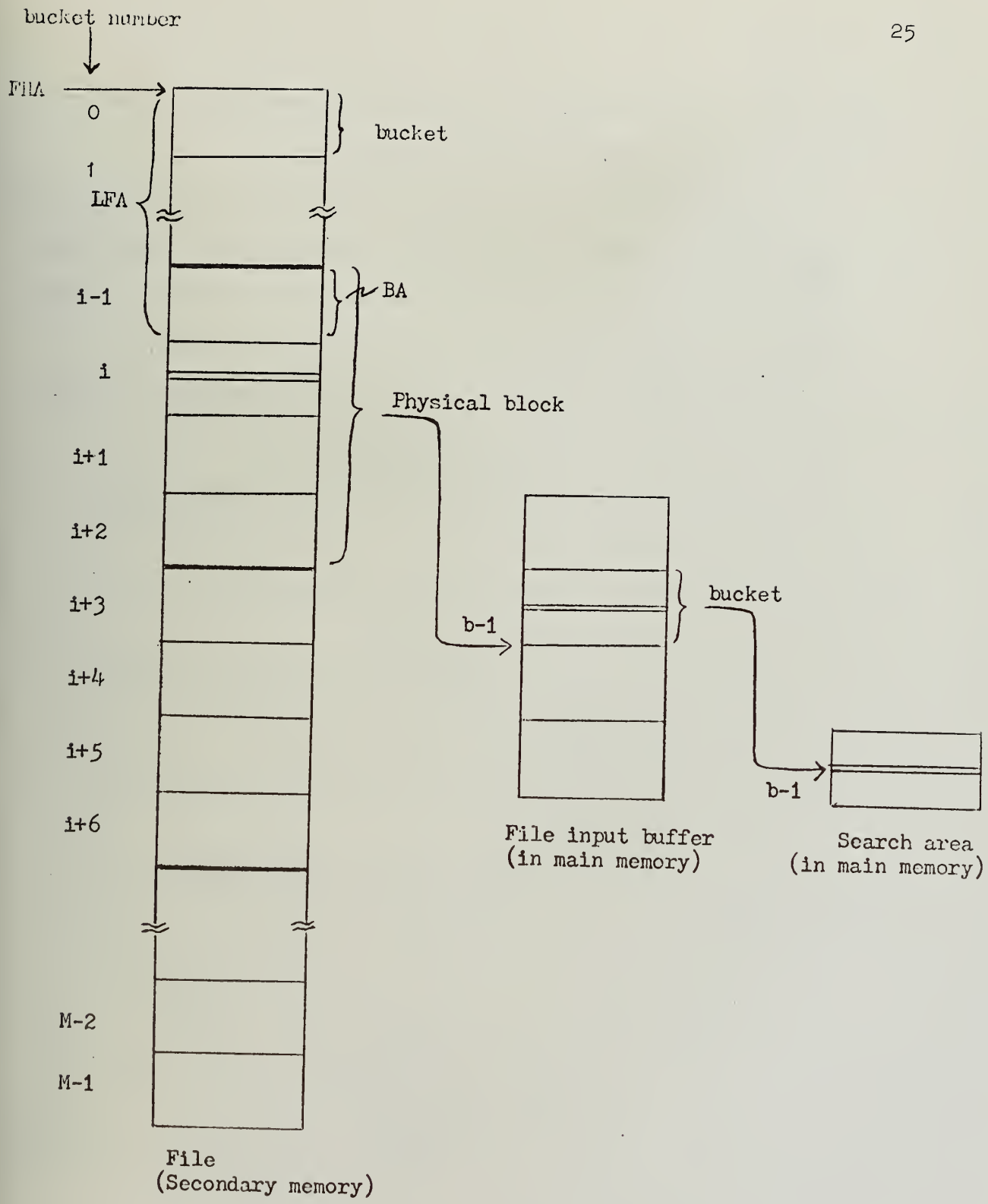


Figure 3.5 Data Structure in Hash External Search Method Environment

search area. After the bucket is moved to the search area, linear search technique is used to find the required record.

The flowchart of the external search routine is shown in Fig. 3.6. When the external search routine is called, argument lists include keys and operation code (access or insert or delete) as parameters. The steps in which the external search routine is performed are as follows:

1. Key-items 'X, Y, Z' are processed by some arithmetic-logical function to produce a single key 'K'.
2. The hash address value 'i' is calculated using a hash function h(K).
3. The bucket i is read from the secondary memory into the search area.
4. The linear probing begins within this search area from the bottom entry of the bucket.
5. If the entry is empty or the key in the entry matches, the probing ends.
6. Otherwise, the probing repeats within a loop until one of the two conditions described is satisfied. While in this probing loop if all entries within a bucket are searched unsuccessfully, another bucket is read-in from the secondary memory. Probing proceeds in this way in a descending address order.
7. If the key-match is the condition to leave the probing loop, next two steps are performed.

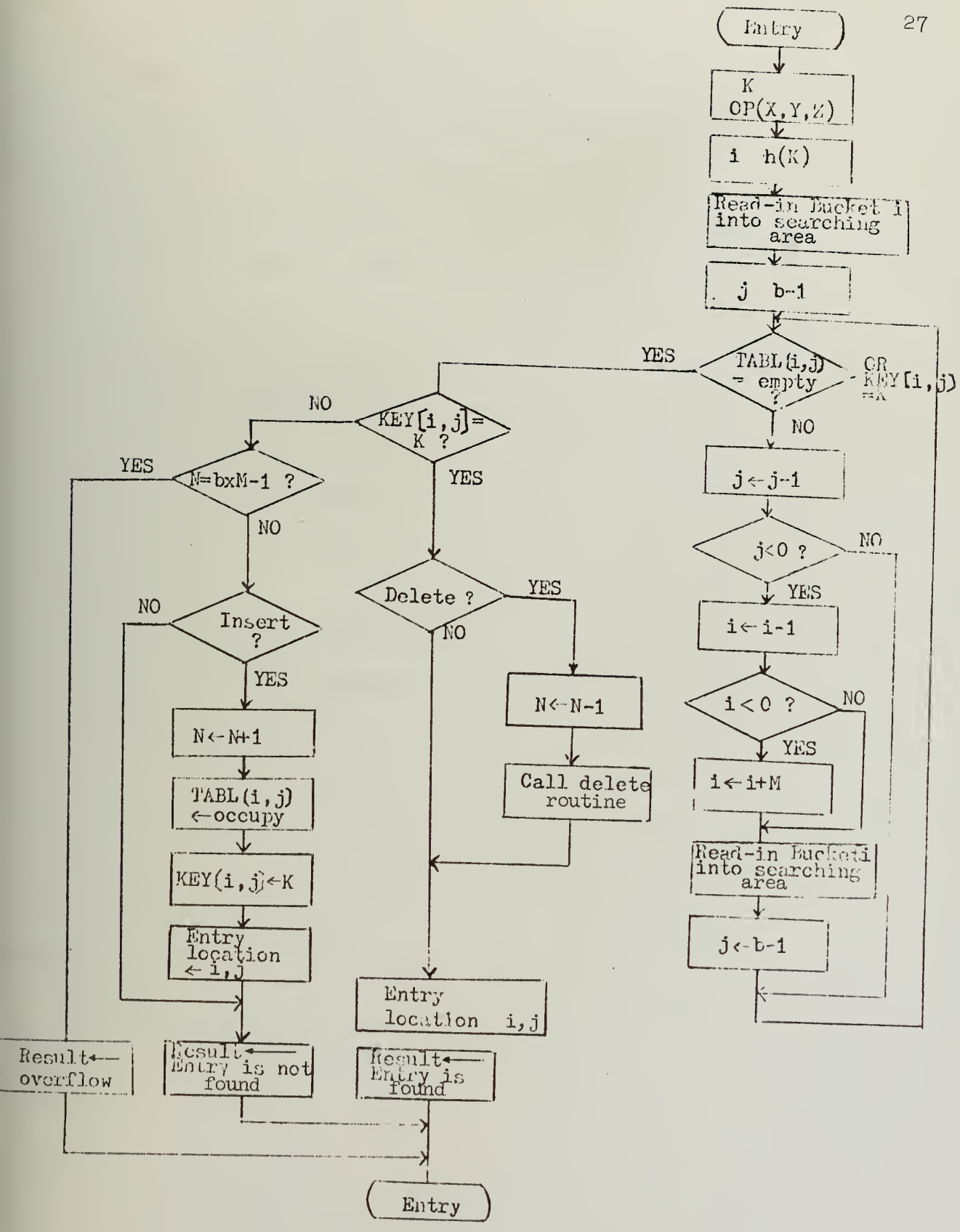
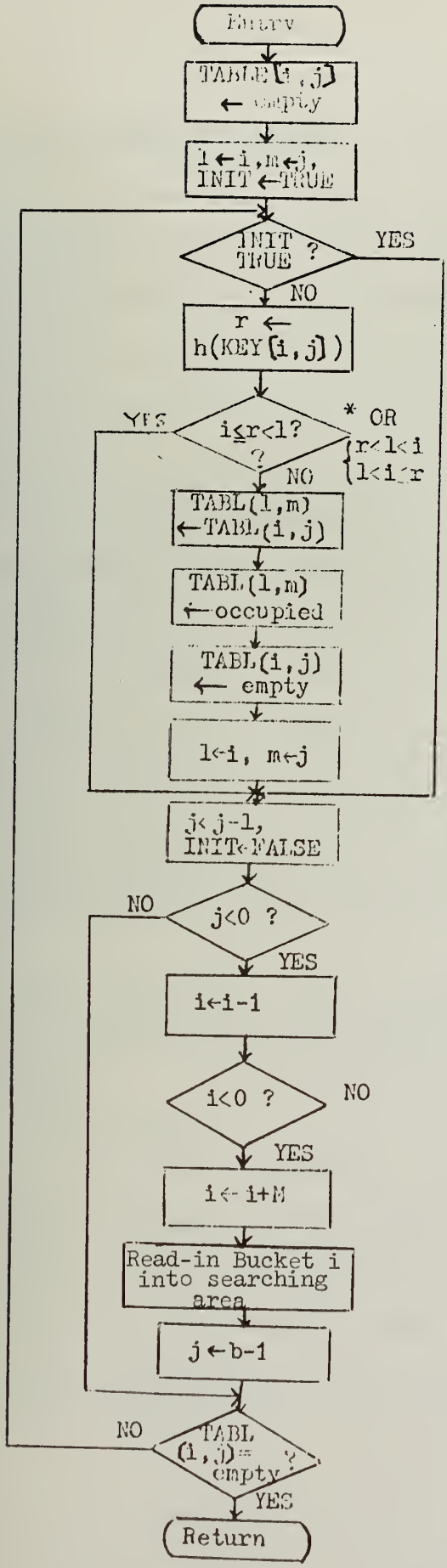


Figure 3.6 External Search Routine

8. If the specified operation is a delte operation, the number of occupied entries 'N' is decreased by one and the delete routine is invoked.
9. The routine returns with the bucket number 'i', the address of entry within the bucket 'j', andthe result code 'entry is found'.
10. However, if 'an empty entry is found' is the condition to leave the probing loop, then one of the next two steps is followed.
11. If the overflow condition is encountered (i.e., $N = b \times M - 1$), the result code 'overflow' is returned and the routine returns.
12. Otherwise, if the specified operation is an insert-operation, 'N' is increased by one and the entry is marked 'occupied', the key is inserted into the entry and the inserted address is returned.
13. The result code 'entry is not found' is returned and the routine returns.

The delete routine is invoked by the external search routine when delete operation is required. This routine is considered to be an extension of external search routine. Fig. 3.7 shows the flowchart of the delete routine. The steps of the delete routine are as follows:

1. The entry located by the external search routine is marked 'empty'.
2. The routine proceeds to examine entries in a descending order. First, the adjacent descending address entry



* Condition is shown schematically below

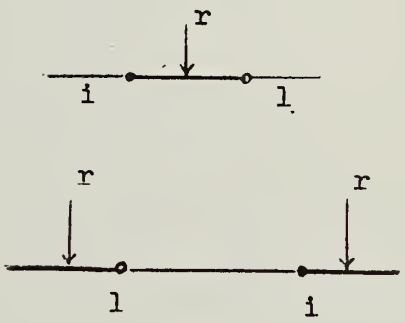


Figure 3.7 Delte Routine

is checked; if it is found to be empty, the routine immediately returns.

3. If the adjacent entry is not empty and this is within the same bucket, the content of this entry is moved downward to the entry originally marked 'empty'; this entry is now marked 'occupied' and the entry just vacated is marked 'empty'.
4. Proceeding in this way, entries within a bucket are moved downward, one at a time; and if after every entry has been moved and an empty entry still has not been encountered, then the next higher bucket needs to be examined.
5. If the key value stored in the empty of the next higher bucket is greater than the key value stored in the entry of the lower bucket last vacated, then the content of the entry in the higher bucket is moved to the entry in the lower bucket.
6. In this way, the operation to search for an empty entry continues proceeding in a descending order until an empty entry met.

This delete routine seems to be complex and require a lot of time if an empty is not formed. However, under normal circumstances it is quite probable that an empty is found within the first bucket if the load factor (the ratio of occupied entries to total entries) is approximately 60% [6]. Hence in general, the search routine will terminate within a short period of time.

3.5 Command Handling Routine

As a conclusion of the file system design, the flow of the command handling routine which handles terminal-command operations discussed in Chapter 2 is presented in this section. Although this routine is an implementation of flow presented in section 2.3.1, it uses all the results of file system design discussed so far in this chapter. This routine uses the following main-memory working areas; these are declared in the following PL/1-like statements:

```
/* FOLLOWING TWO DATA AREAS ARE I/O AREAS                              */
1  INDATA
   2 COMMAND  CHAR(7),
   2 SUBCOM   CHAR(7),
   2 FLTNUM   CHAR(3),
   2 FLTDATE
     MONTH    CHAR(3),
     DAY      CHAR(2),
     YEAR     CHAR(2),
   2 SEAT     CHAR(3),
   2 NAME     CHAR(20),
   2 ORIGIN   CHAR(3),
   2 DESTIN   CHAR(3),
   2 DATA     CHAR(100);

1  OUTDATA    CHAR(100);

/* FOLLOWING THREE DATA AREAS ARE WORKING AREAS                        */
1  FLTDATA,                          /* FLTLIST IS READ-IN           */
   2 FLIGHT,
     3 FRSEAT CHAR(3),               /* COUNT OF FREE SEATS          */
     3 CHAIN  BIN(32);               /* POINTER TO PSRLIST           */

1  PSRDATA,                          /* WHERE PSRLIST IS READ-IN     */
   2 PVSEAT   CHAR(3),               /* COUNT OF RESERVED SEAT       */
   2 PHONE    CHAR(10),
   2 ADDRESS,
     3 NUM    CHAR(6),
     3 STREET CHAR(15),
     3 CITY   CHAR(15),
     3 STATE  CHAR(4),
   2 RVSTATE  CHAR(9),               /* RESERVED/CONFIRMED           */
```

```
1 INFODATA(*)  CTL,                    /* WHERE INFORLIST IS READ-IN    */
  2 ORIGIN,
    3 TIME     CHAR(5),
  2 DESTIN,
    3 TIME     CHAR(5),
  2 FLTNUM     CHAR(3),
  2 OPDATE     CHAR(5),
  2 EQUIP      CHAR(5),
  2 FARE       CHAR(6);
```

The flowchart of command handling routine is shown in Figs. 3-8-Fig. 3.12. Note that STRING(INDATA) is a string manipulation operation to edit and move the INDATA to the PSRDATA area.

3.6 Summary

The file system described in this chapter was designed to satisfy the requirements of the reservation system described in Chapter 2. The organization of file system was discussed. Then the logical file system was designed to meet the requirements of this system and the hash addressing search routine was presented. Finally the command handling routine was presented as a conclusion of the file system design.

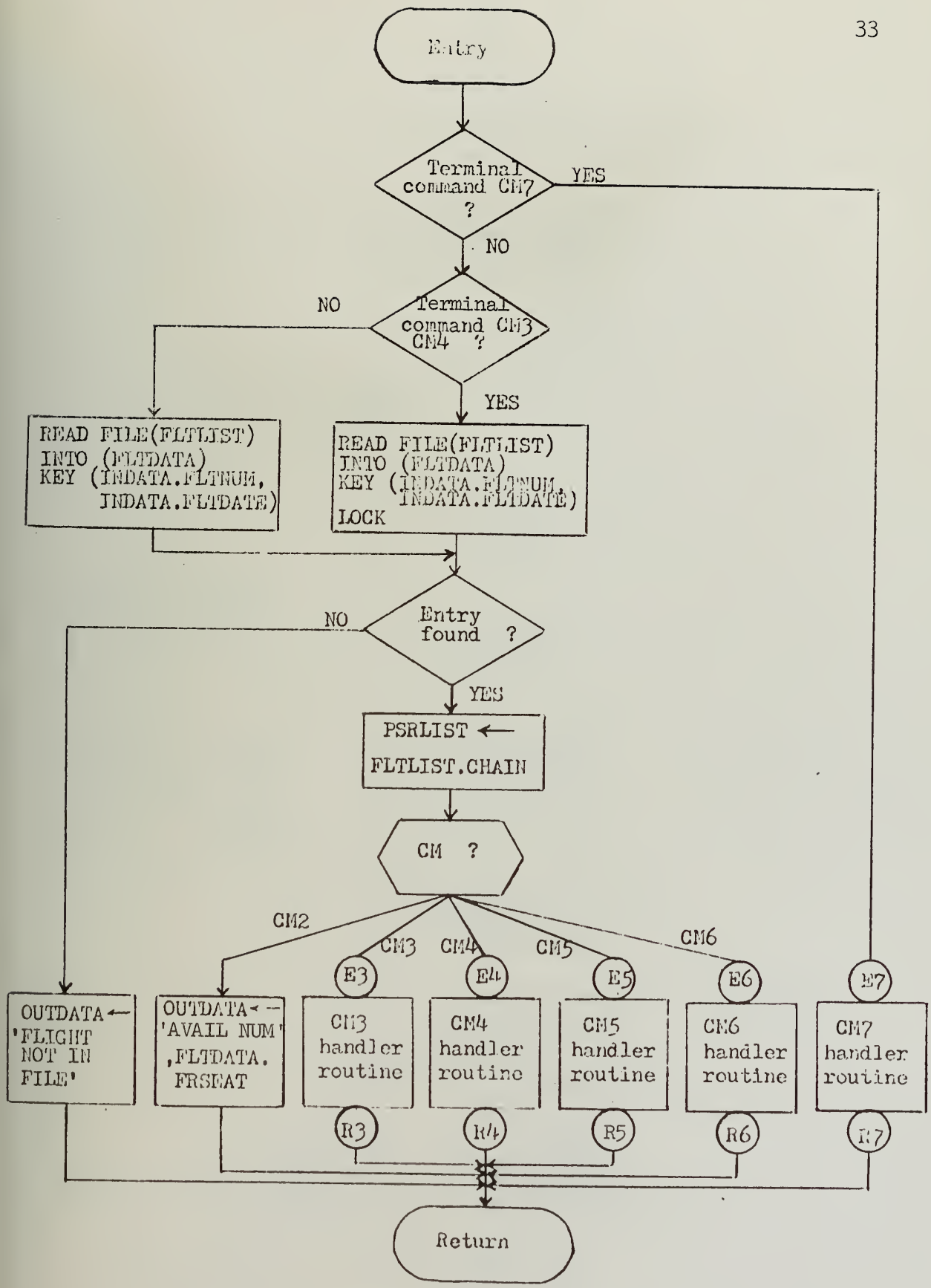


Figure 3.8 Flow of Command Handler Routine (1)

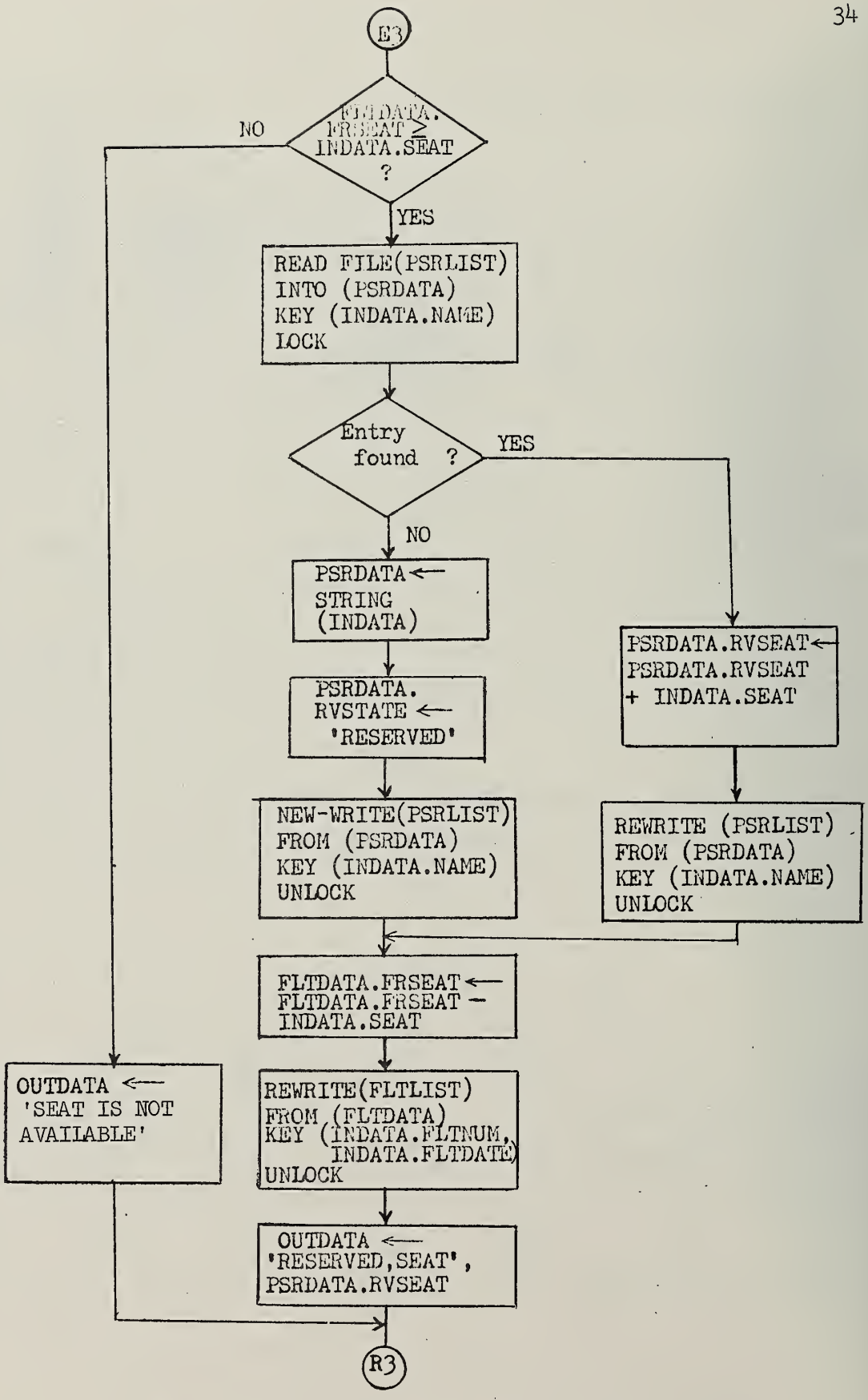


Figure 3.9 Flow of Command Handling Routine (2)
(CM3-Reservation)

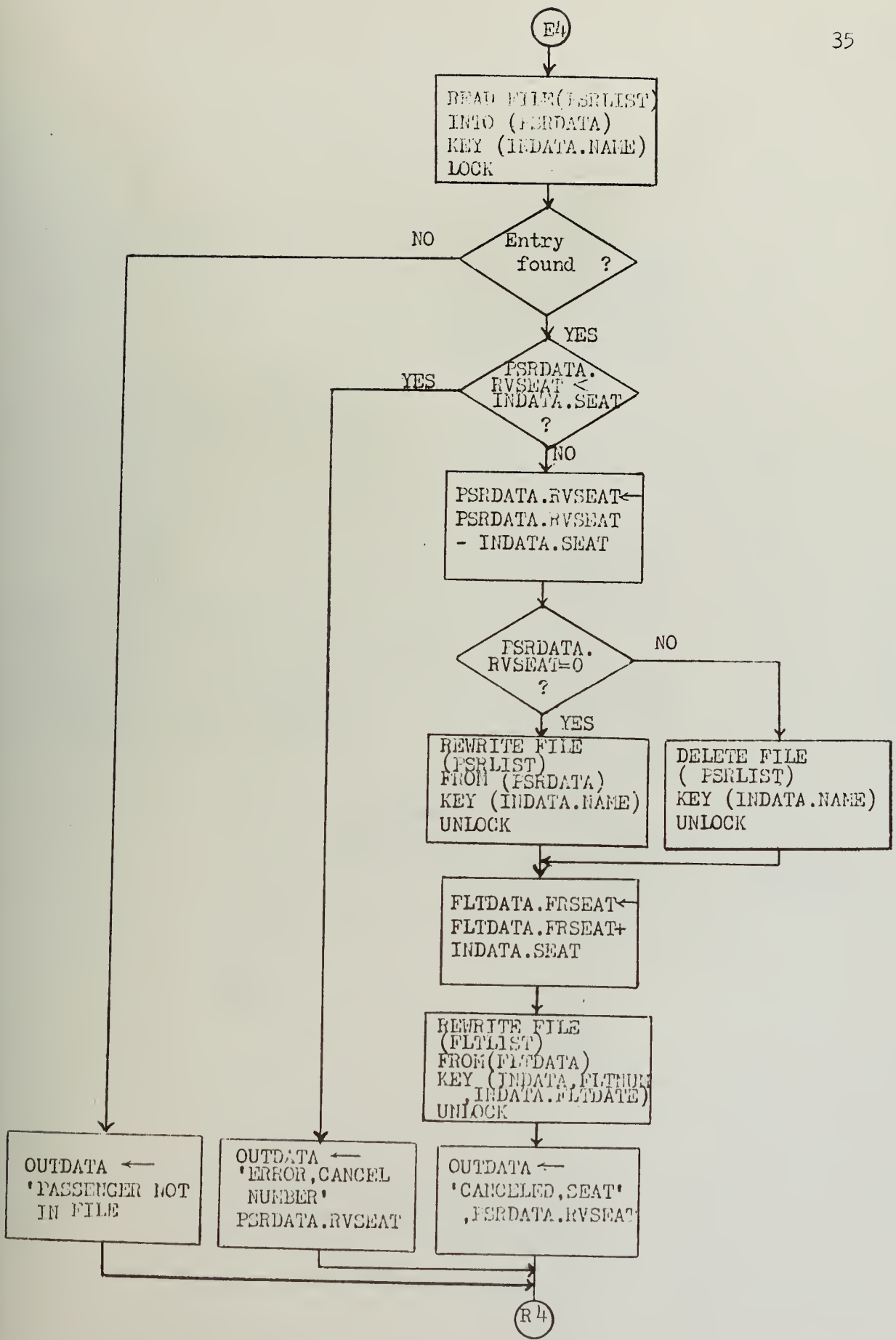


Figure 3.10 Flow of Command Handling Routine (3)
(CM4 - Cancellation)

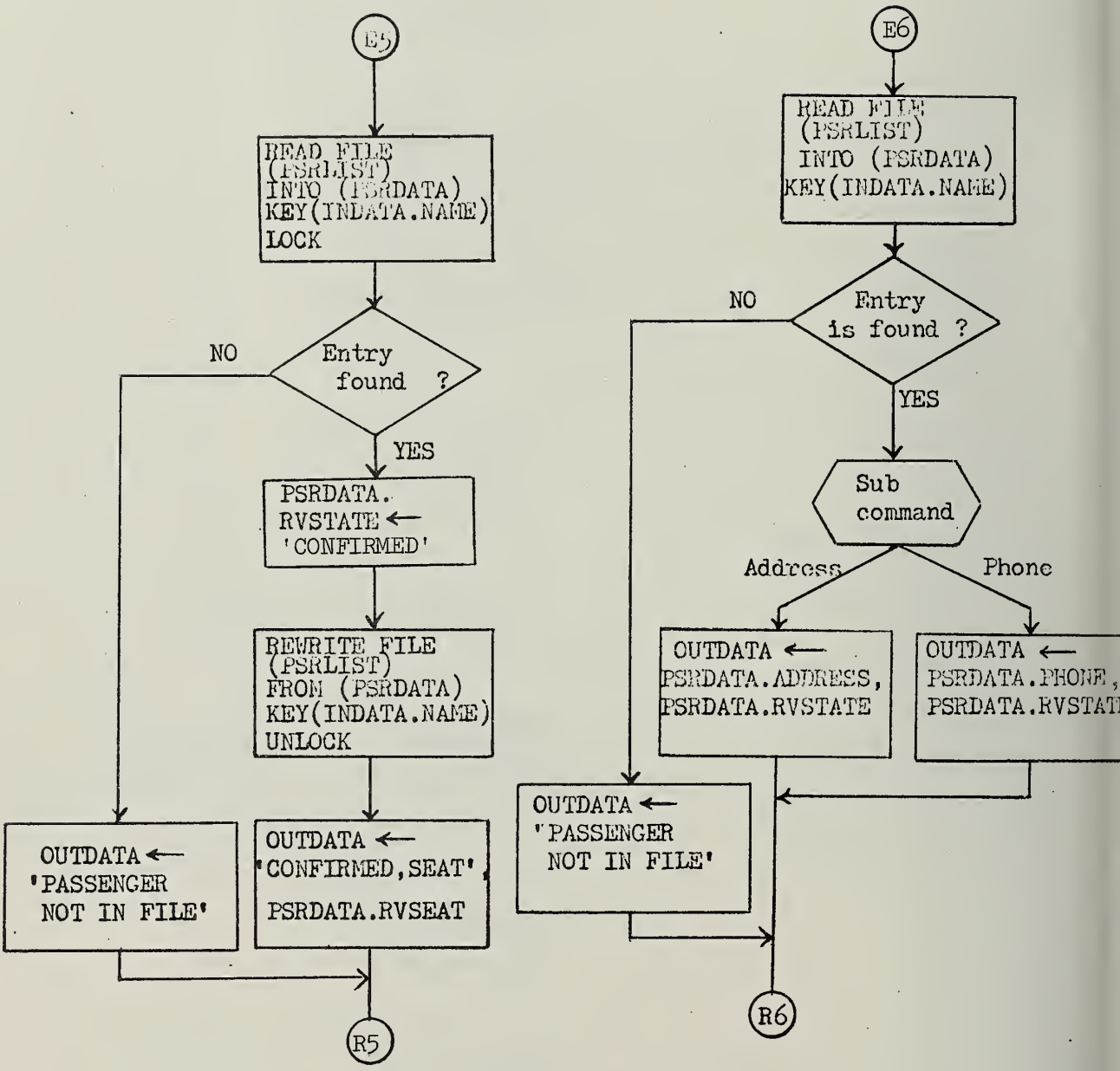


Figure 3.11 Flow of Command Handling Routine (4)
(CM5-Confirmation, CM6-Passenger's
Information)

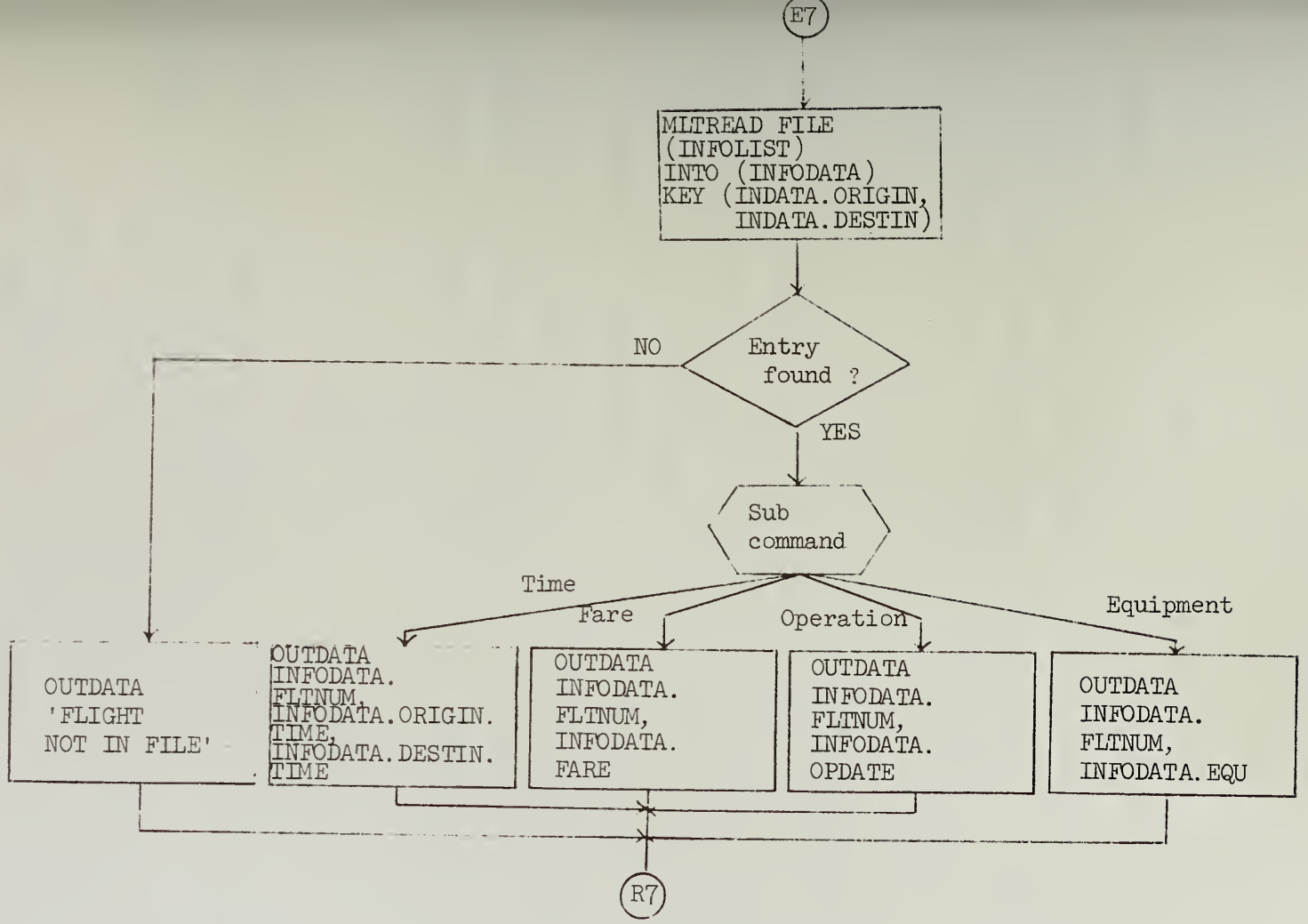


Figure 3.12 Flow of Command Handling Routine (5)
(CM7-Flight Information)

4. CENTRAL COMPUTER DESIGN

In this chapter, the hardware architecture and the operating systems for the reservation system are described. The central computer in this system consists of network of functionally dedicated processors. The network has a multi-level cluster structure where any node in the network may be one special purpose processor or a sub-network of several processors.

The operating system is designed based on the functional process concept [7,8]. Other features of the operating system include: (i) hierarchical level of monitor (kernel), (ii) process synchronization by semaphores and (iii) inter-process communication via message buffer.

The requirement of the central computer for the reservation system is summarized in Section 4.1. Its detailed design is presented in Section 4.2 and Section 4.3. The hardware architecture of the cluster network with dedicated processors and the operating system are described in Section 4.4.

4.1 Summary of Operation

When a terminal command is processed, it is already brought in core. However, the terminal command is sent in bit serial manner and it is sometimes required to edit the command such that the command just sent is merged with the commands already in core. (For example, a command which asks the new reservation can be used without flight

number and date after sending the available seat inquiry command (see Section 2.2) because such information is already included in the available seat inquiry command.) For this reason, all command messages are stored in the secondary storage called Message Assembly File (MAF). The sequence of events in handling the command message from the terminal is described in Fig. 4.1.

The required processing of terminal command in the line input operation is mainly determined by the configuration of communication line and the method of sending the terminal command. To be specific the following conditions are assumed in this reservation system: (i) a point-to-point line configuration with 40 terminals is used, (ii) all terminals are teletype-like devices, (iii) sending format is a 11-bit start-stop code, and (iv) the initiation of the line input operation is caused by external interrupt. Step-by-step reference in the line input operation is shown in Fig. 4.2.

Scheduling operation and edit operation are described, respectively, in Figs. 4.3 and 4.4. Round-robin scheduling policy is used in this system. This system allows two types of jobs, interactive job and batch job to run simultaneously. The interactive jobs are assigned higher priority.

The command handling operation was discussed in section 3.5 (see Figs. 3.8 - 3.12.

The line output operation which sends back the reply to the command sending terminal is described in Fig. 4.5.

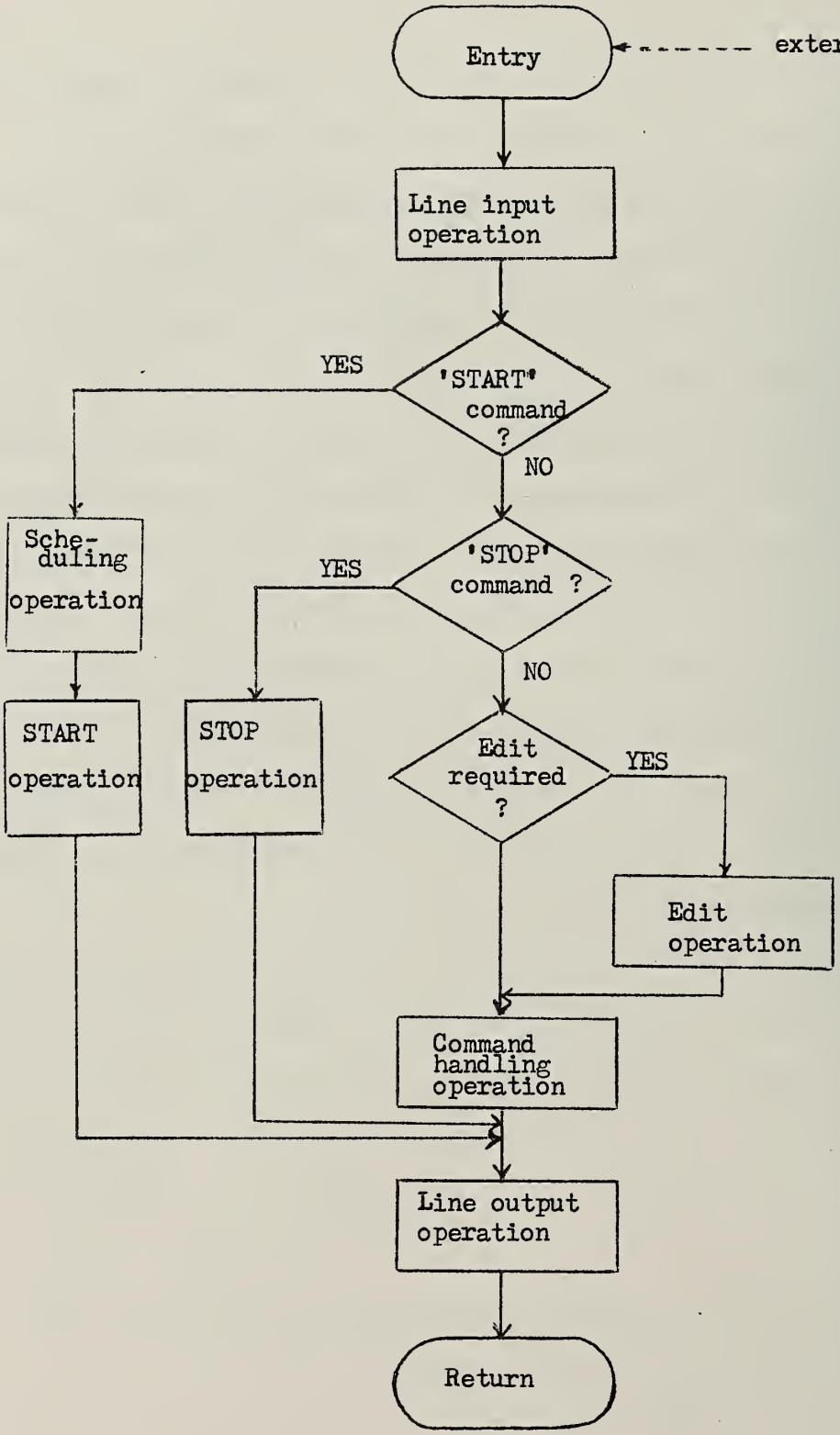


Figure 4.1 Sequence of Events for the Reservation System

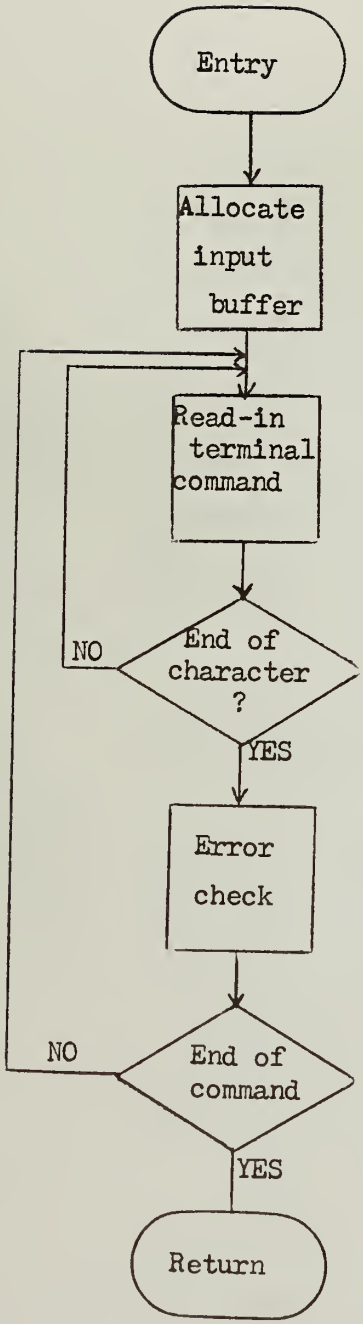


Figure 4.2 Line Input Operation

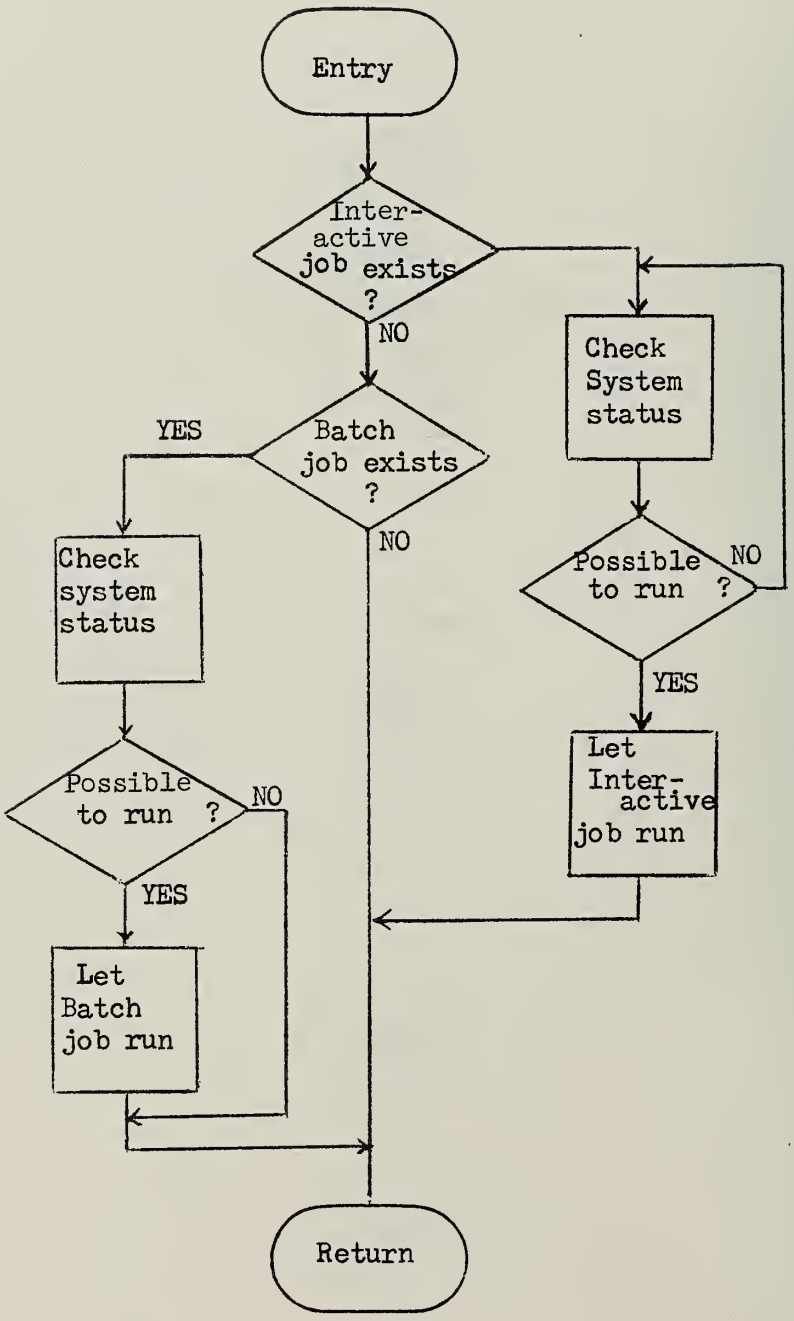


Figure 4.3 Scheduler Operation

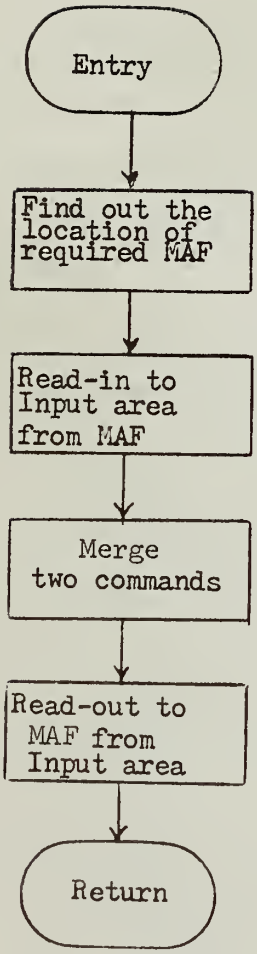

Entry
Find out the location of required MAF
Read-in to Input area from MAF
Merge two commands
Read-out to MAF from Input area
Return

Figure 4.4 Edit Operation

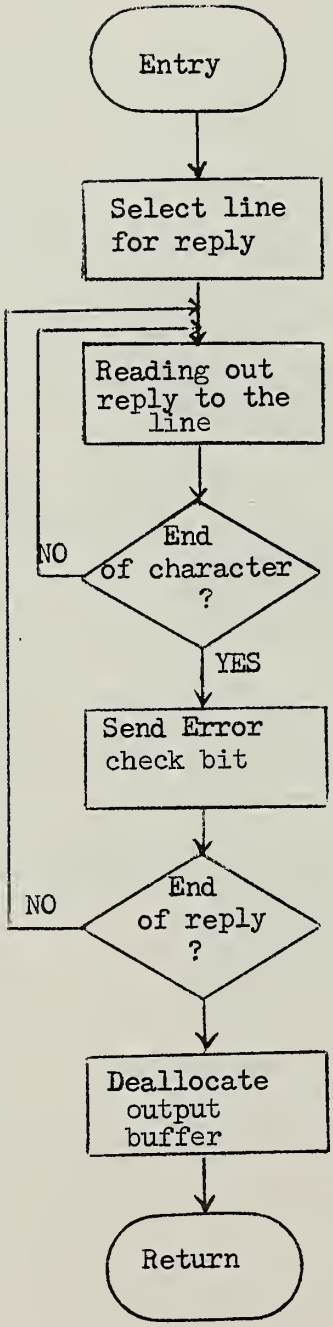


Figure 4.5 Line Output Operation

4.2 System Design

To design a cluster network of functional dedicated processors, the system is decomposed into cooperative processes.* We discuss here the different processes and the format of messages for process communication. Then the structure of the network with dedicated nodes and required communication path between them are determined.

4.2.1 System Decomposition Rules

To decompose the system into processes, the following rules are observed. The operations performed in a process must consist of self-contained functions. Such block of operations must be logically deep enough so as to decrease an overhead loss. The decomposition must be done in such a way that the length of inter-process message be as short as possible. Moreover, the system is decomposed into a family tree of processes and only the parent process has the right to create and destroy a child process. Within the family tree of processes a process can only communicate with the parent process, the child processes, and the brother processes.

4.2.2 Decomposition of Reservation System into Processes

A family tree of processes for the reservation system is generated where each node corresponds to a process and each link corresponds to a create and destruct relation between processes.

---

*A process is a computation in which operations are carried out strictly one at a time [9]. Hence, a process usually runs with other processes concurrently.

Fig. 4.6 shows the family tree of processes for the reservation system. The root of the tree consists of the initializer process which is loaded by the Initial Program Loader (IPL). The other processes are:

1. Line input process: Read in the terminal command into the input buffer area.
2. Interactive job handler process: Control the flow of interactive processing job stream.
3. Line output process: Send out the answer from the output buffer area.
4. Scheduler process: Monitor the overall system status.
5. Command handling process: Execute the command job.
6. Edit process: Edit the command.
7. File management process: Handle the file management system.
8. External search process: Search for the logical record in a file.
9. Probe delete process: Auxiliary process for record search.
10. File input process: Read in from secondary memory to file buffer area.
11. File output process: Read out from file buffer area to secondary memory.

The initializer process creates four types of child processes: the line input process, the line output process, the interactive job handler process and the scheduler process. (Former three types of processes are created corresponding to the terminal of this reservation system.)

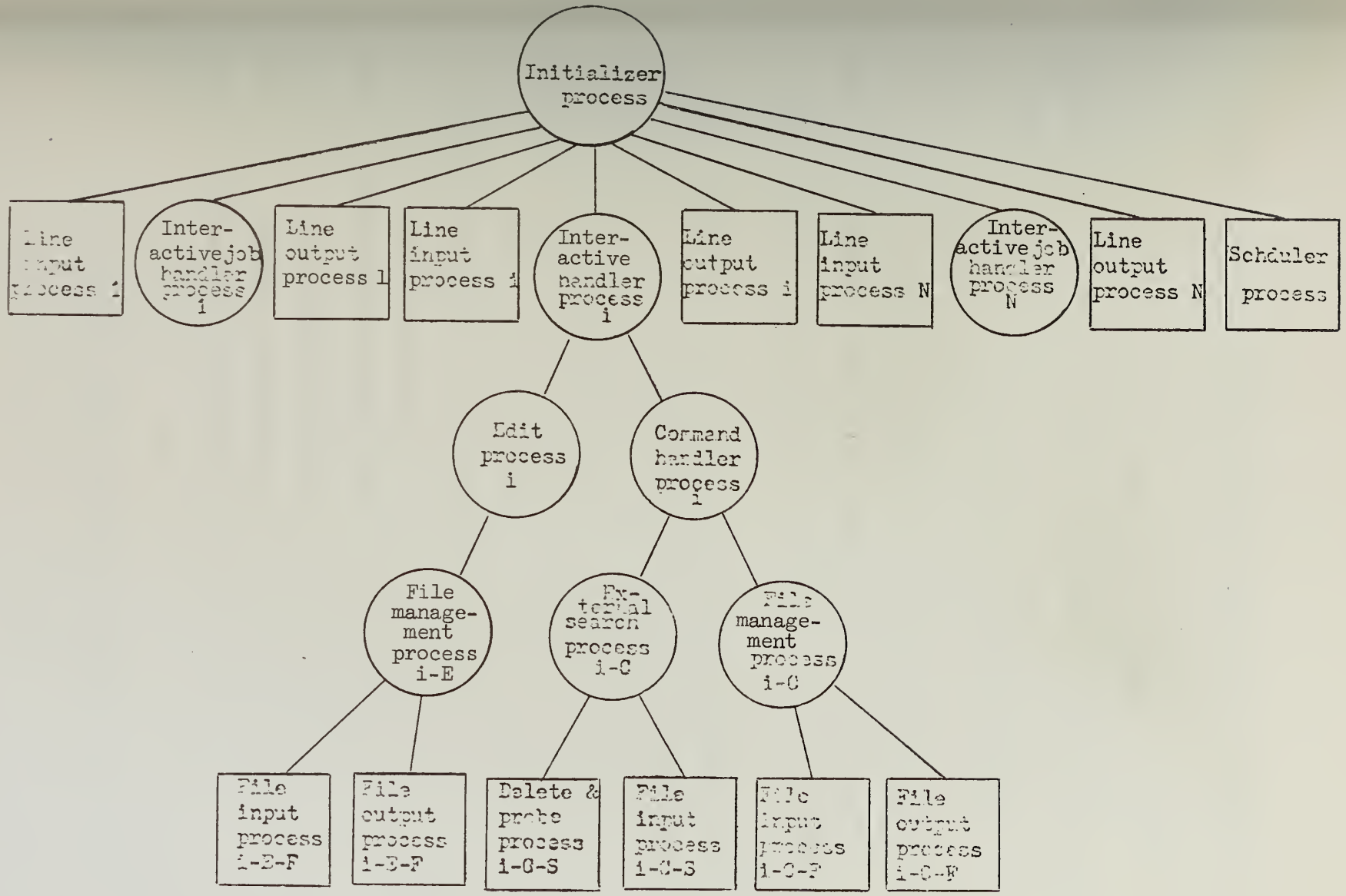


Figure 4.6 Family Tree of Process in the Central Computer

When the system is initialized, only these five process types exist in the system.

When the terminal command 'START'--which is a kind of control command--is sent from a terminal, the remaining processes are dynamically created by their parent processes orderly. When the terminal command 'STOP' is sent, these processes are destroyed.

4.2.3 Operation Flow by Process Cooperation

After a transaction is started, the flow of operation by process cooperation following the introduction of the terminal command is as follows:

1. Terminal command request causes external interrupt to the system. The line input process is initiated. It does the read-in of the terminal command into the input buffer area. On the completion of the input operation, it sends a message to the interactive job handler process.
2. The interactive job handler process, receiving a message from the line input process, sends a message to the edit process if the terminal command needs editing.
3. The edit process does the editing operation by cooperating with the file management process and sends back a message to the interactive job handler process.
4. The interactive job handler process then sends a message to the command handler process.

5. The command handler process together with the external search process and the file management process does necessary operations according to the type of terminal command and sends back a message to the interactive job handler process.
6. The interactive job handler process in turn sends a message to the line output process.
7. The line output process does the read-out of the reply from the output buffer to the line.

4.3 Reservation System Processes

In this section the flow of each process is described in detail. We assume that the flow of operation is implemented by the microprogram code in the processor dedicated to the process. Hence, the process flow can be considered to be the flow of the microprogram code for each specific function of the reservation system.

4.3.1 Line Input Process

The flow of the line input process is shown in Fig. 4.7. This process is an interrupt driven process. This process waits for an external interrupt by doing a P-operation on the Interrupt Wait Semaphore (IWS) in the Line Control Block (LCB). When an external interrupt occurs, the External Interrupt Handler Routine (EIH) performs a V-operation on this semaphore. Then the line input process is initiated It first allocates an input buffer, then reads the line input serially

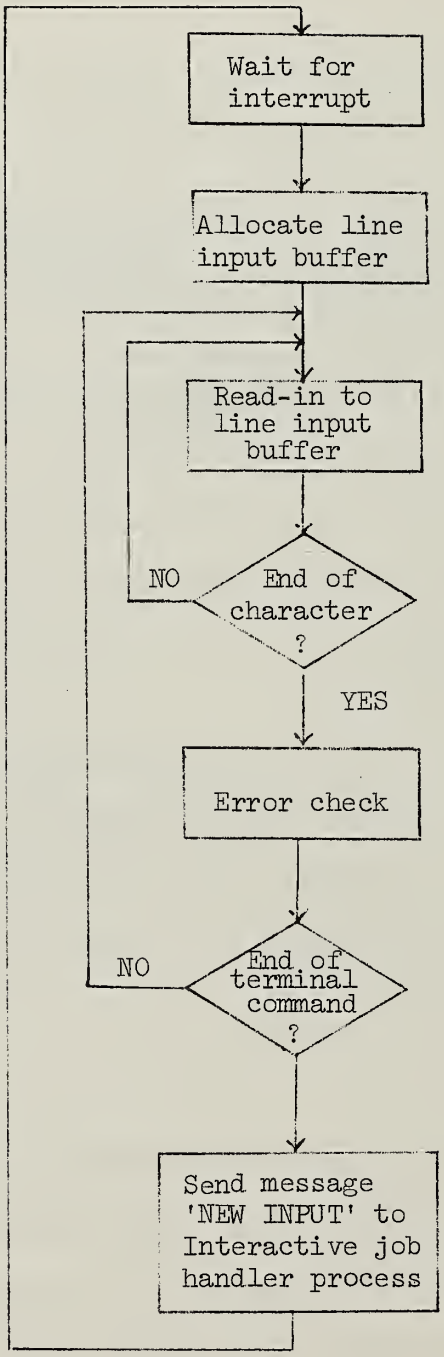


Figure 4.7 Flow of Line Input Process

into the buffer, checking for errors and, on completion of the input operation, a message which indicates a new input is now in buffer is sent to the interactive job handler process.

After finishing all these operations, it again performs a P-operation on the IWS and goes into the "waiting for interrupt" state.

4.3.2 Interactive Job Handler Process

The flow of the interactive job handler process is shown in Fig. 4.8 and Fig. 4.9. This process is the interactive job monitor process for controlling the stream of other processes. It receives a message from the line input process about the new arrival of terminal command. First, it allocates a working area for the terminal command. Then it moves the input command from the input buffer to the working area and deallocates the input buffer.

If the input command is the control command 'START', it creates the Interactive Job Control Block (IJCB) and puts the IJCB into the Interactive Job Queue (IJQ). The scheduler process will decide whether this interactive job is allowed to enter the system. The interactive job handler process waits for a message from the scheduler process. This process then decreases the items in the system resource table, creates, then initiates the edit and command handling processes, and then generates the reply which acknowledges 'START'.

If the input command is the control command 'STOP', this process stops and destroys the edit and command handler processes, updates the resource items in the resource table, removes the IJCB from the IJQ, and then creates the reply which acknowledges 'STOP'.

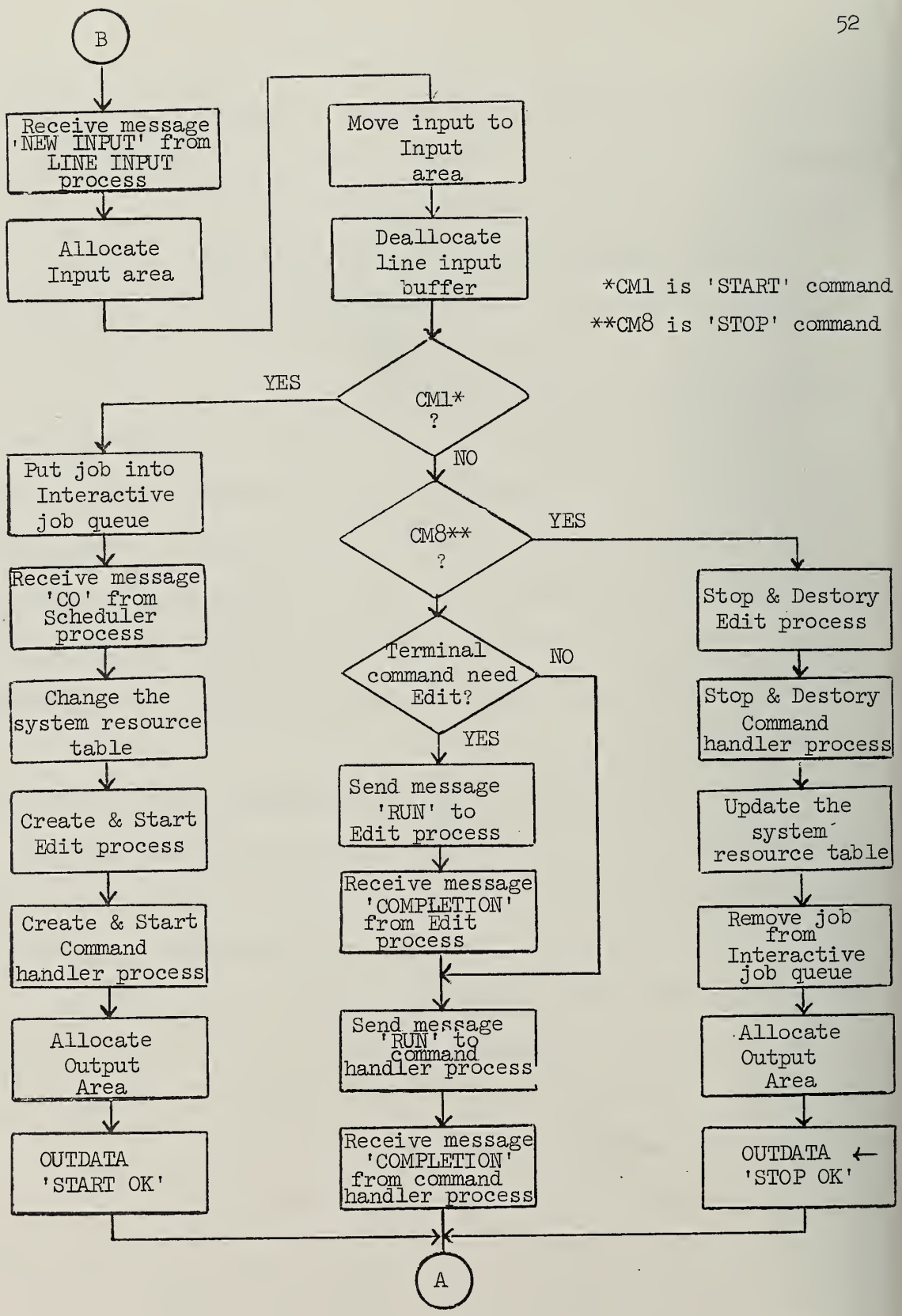

B
Receive message 'NEW INPUT' from LINE INPUT process
Allocate Input area
Move input to Input area
Deallocate line input buffer
*CM1 is 'START' command
**CM8 is 'STOP' command
CM1* ?
YES
NO
Put job into Interactive job queue
CM8** ?
YES
Stop & Destory Edit process
Receive message 'CO' from Scheduler process
Terminal command need Edit?
NO
YES
Stop & Destory Command handler process
Change the system resource table
Send message 'RUN' to Edit process
Update the system resource table
Create & Start Edit process
Receive message 'COMPLETION' from Edit process
Remove job from Interactive job queue
Create & Start Command handler process
Send message 'RUN' to command handler process
Allocate Output Area
Allocate Output Area
Receive message 'COMPLETION' from command handler process
OUTDATA 'START OK'
OUTDATA 'STOP OK'
A

Figure 4.8 Flow of Interactive Job Handler Process (1)

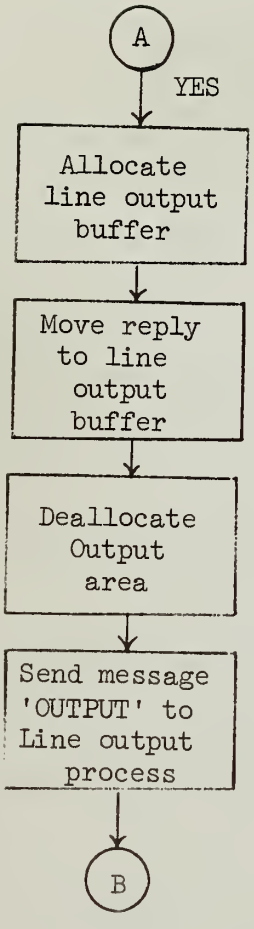


Figure 4.9 Flow of Interactive Job Handler Process (2)

If the input command is any other type of terminal command, it first checks whether the terminal command needs editing, and if it does, this process sends a message to the edit process. After receiving a "completion" message from the edit process it then sends a message to the command handler process and waits for a "completion" message.

After all operations discussed above are finished, this process allocates an output buffer, moves the reply to the output buffer, deallocates the working area and sends a message to the line output buffer. The message shows the reply to the terminal which is now in the output buffer.

4.3.3 Line Output Process

The flow of the line output process is shown in Fig. 4.10. This process performs the operation of returning replies to the terminal. This process first waits for a "run" message from the interactive job handler process.

After it is initiated, it sends the reply to the terminal bit-serially until it encounters the "end of answer file." Then it deallocates the output buffer and informs the interactive job handler process that the output buffer is available.

4.3.4 Scheduler Process

The flow of the scheduler process is shown in Fig. 4.11. This process monitors the status of all system resources and determines the policy for job entry into the system. According to the scheduling policy employed in this system high priority is given to the interactive jobs over the batch jobs. Hence, two job queues are structured for each job class.

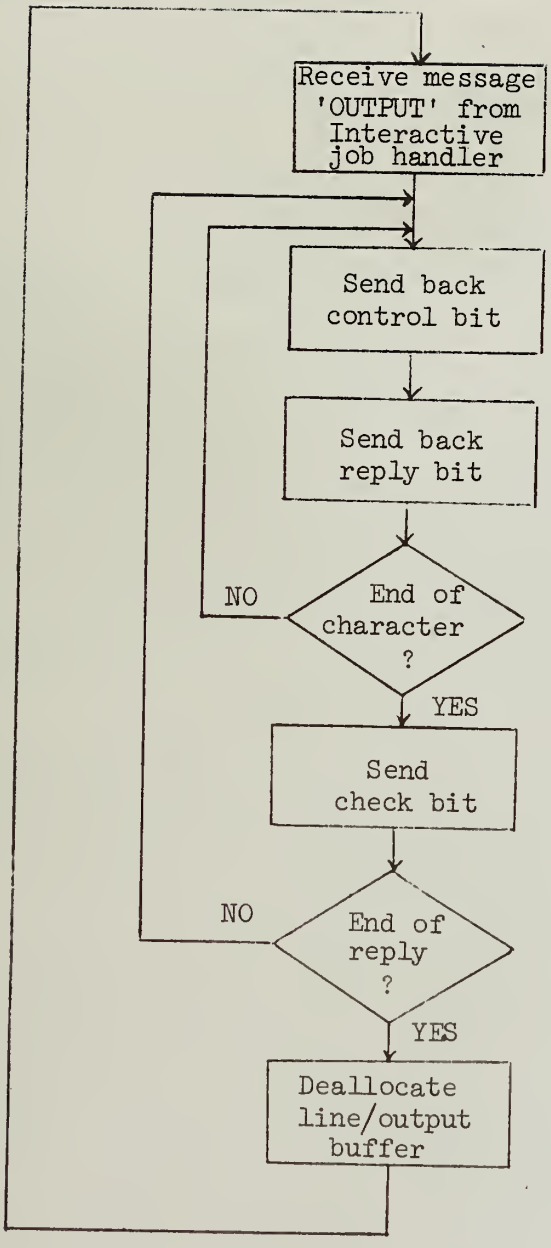


Figure 4.10 Flow of Line Output Process

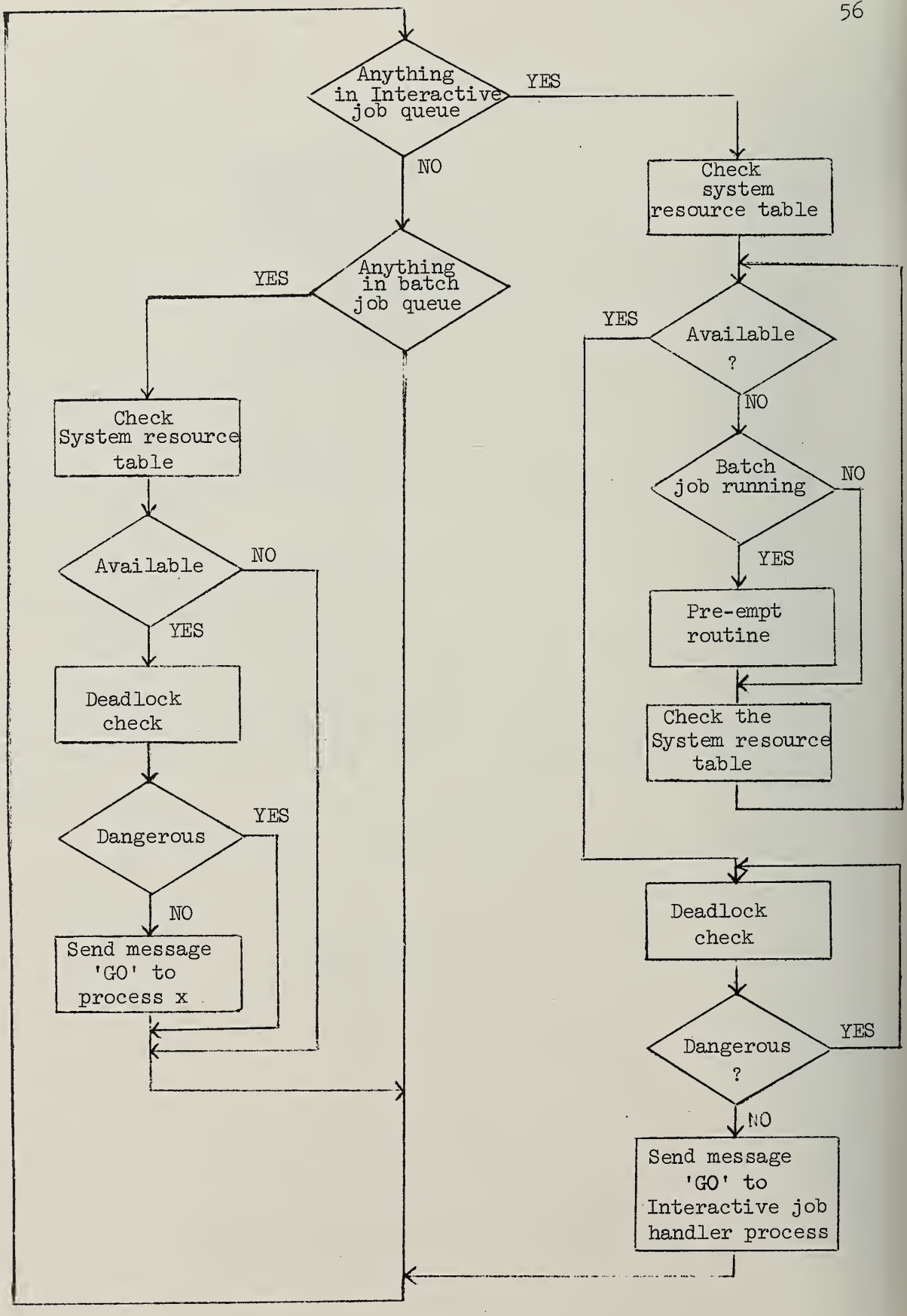


Figure 4.11 Flow of Scheduler Routine

This process first examines the interactive Job Queue (IJQ). If any interactive job is waiting it checks the system resource table to see if the necessary resources are available and then checks possibility of creating a deadlock. Then it sends a "go" message to the interactive job handler process. In this case, if the system resources are not found to be available, it performs a pre-empt operation on a batch job and again checks the availability of system resources.

However, if there is no job in the IJQ it then examines the Batch Job Queue (BJQ). It also checks the resource availability and the deadlock possibility, and, if it finds the batch job cannot run, process flow just returns to the main scanning loop and again tries to find an interactive job in the IJQ.

4.3.5 Command Handler Process

The flow of the command handler process is shown in Fig. 4.12. The actual processing of terminal commands is performed in this process. The detailed operation of this process was discussed in section 3.5 (see Fig. 3.8-3.12). The flow of this process is a version of operation flow based on process concept.

When this process is created, this process in turn creates and initiates two processes: the file management and external search processes.

It then waits for a run message from the interactive job handler process. After receiving a message, it performs command handling operations according to the content of the terminal command.

If the terminal command is CM7 then the routine to handle CM7 is entered.

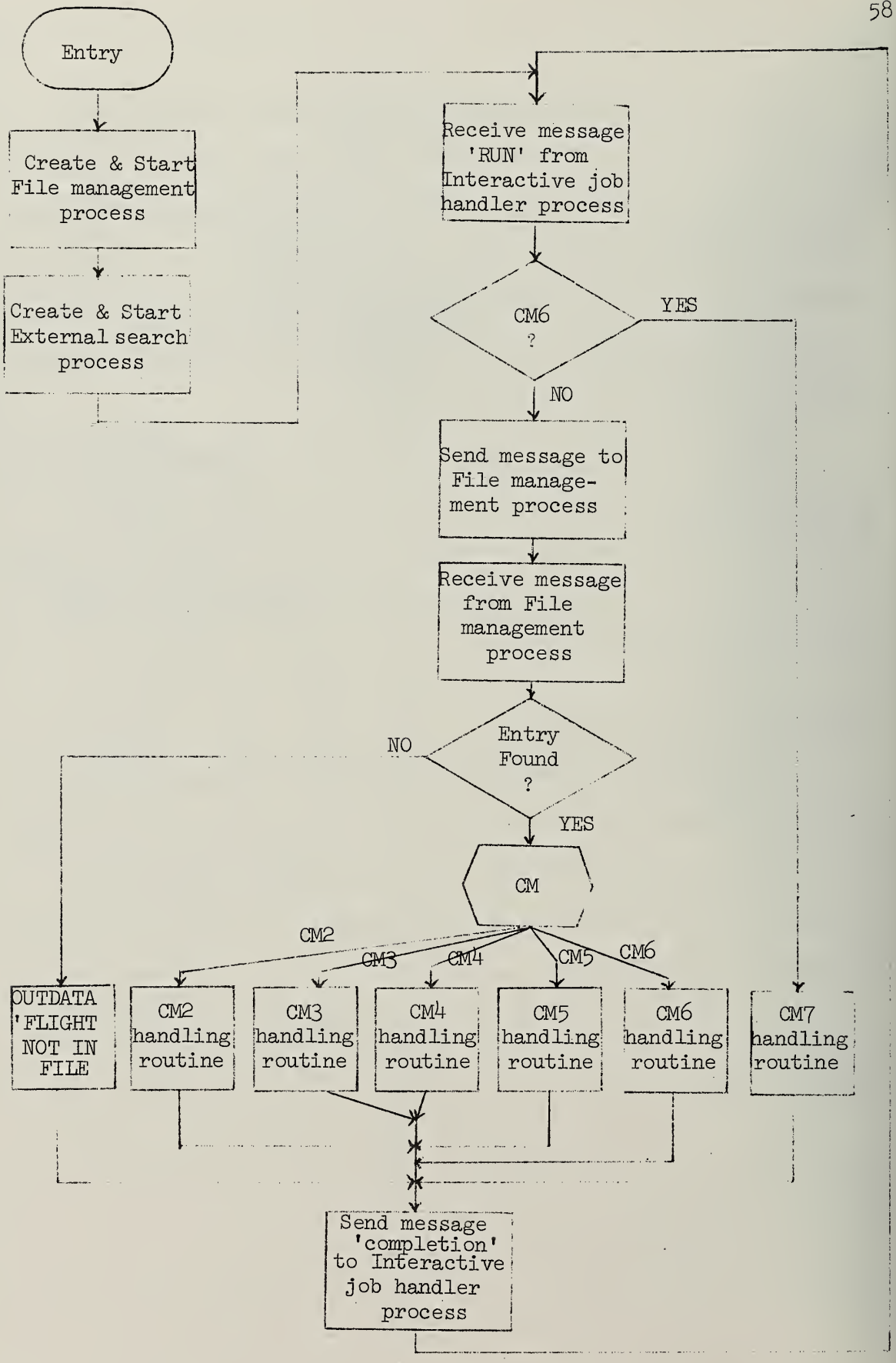


Figure 4.12 Flow of Command Handler Process

If the terminal command is another type of command, it first sends a "run" message to the file management process and waits for a "completion" message from the same process. After checking that the entry of the file has been found, it enters the corresponding command handling subroutine according to the type of terminal command.

After finishing each subroutine operation, it sends a "completion" message to the interactive job handler process.

4.3.6 Edit Process

Fig. 4.13 shows the flow of the edit process. This process edits a terminal command.

When this process is created it also creates the file management process and then waits for a "run" message from the interactive job handler process. After receiving a "run" message, it sends a message to the file management process. This message requires the file management process to read-in the command file which is in a Command Assembly Área (CAA).

This process eventually receives a "completion" message. Then the edit routine is entered. This routine merges the newly-arrived command with the command which is already stored in the CAA.

It then sends a "completion" message to the interactive job handler process and a message to the file management process to read-out the editing result to the CAA.

After that, it enters a state of waiting for a "run" message.

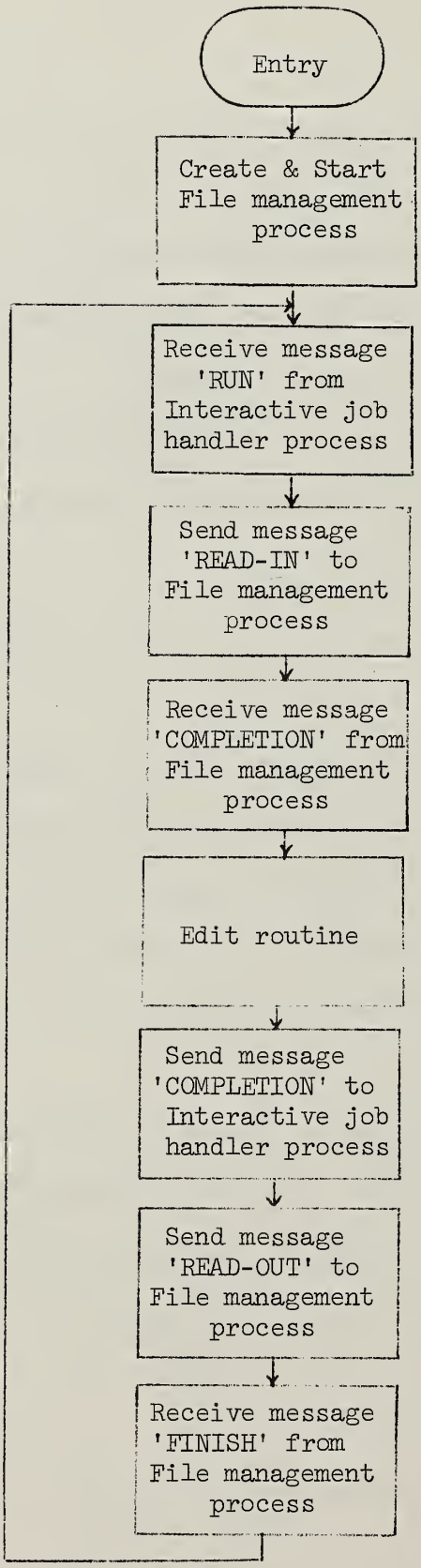

Entry
Create & Start File management process
Receive message 'RUN' from Interactive job handler process
Send message 'READ-IN' to File management process
Receive message 'COMPLETION' from File management process
Edit routine
Send message 'COMPLETION' to Interactive job handler process
Send message 'READ-OUT' to File management process
Receive message 'FINISH' from File management process

Figure 4.13 Flow of Edit Process

4.3.7 File Management Process

The flow of the file management process is shown in Fig. 4.14 and 4.15. This process performs the file management operation which was fully discussed in section 3.3.

When this process is created, it creates two file processes: file input process and file output process. This process waits for a "run" message from the command handler process or the edit process depending on the creater (parent process) of this process. If the message includes a symbolic file name, the Symbolic File System (SFS) and the Basic File System (BFS) routines are entered to obtain the location of the file (see section 3.3).

After obtaining the location of the file, if file-lock is required, it performs a P-operation on the file semaphore which is included in the Unit Control Block (UCB) of the file. This results in a locking of this file and the inhibiting of access of other processes to this file. If file-lock is not required, it just checks the value of the file semaphore and if the value is not positive (i.e., this file is locked) this process enters the waiting state.

After the lock operation, if the searching of record items is required (i.e., if key is given within the message), this process sends a message to the external search process and waits for an answer message.

However, if the requirement is just the read-in or the read-out of an entire file, this process performs the file I/O operation with the cooperation of the file input and output processes.

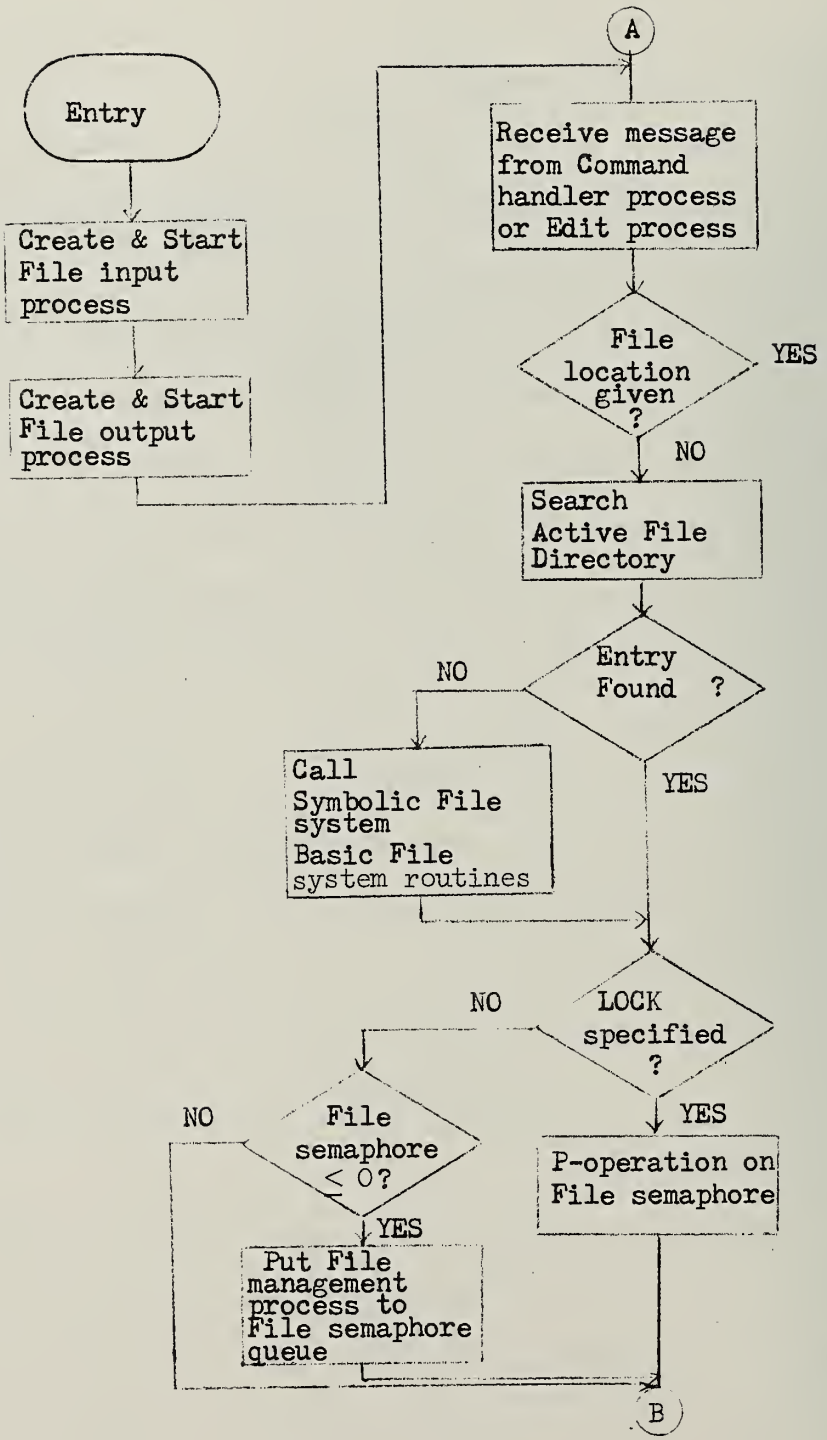


Figure 4.14 Flow of File Management Process (1)

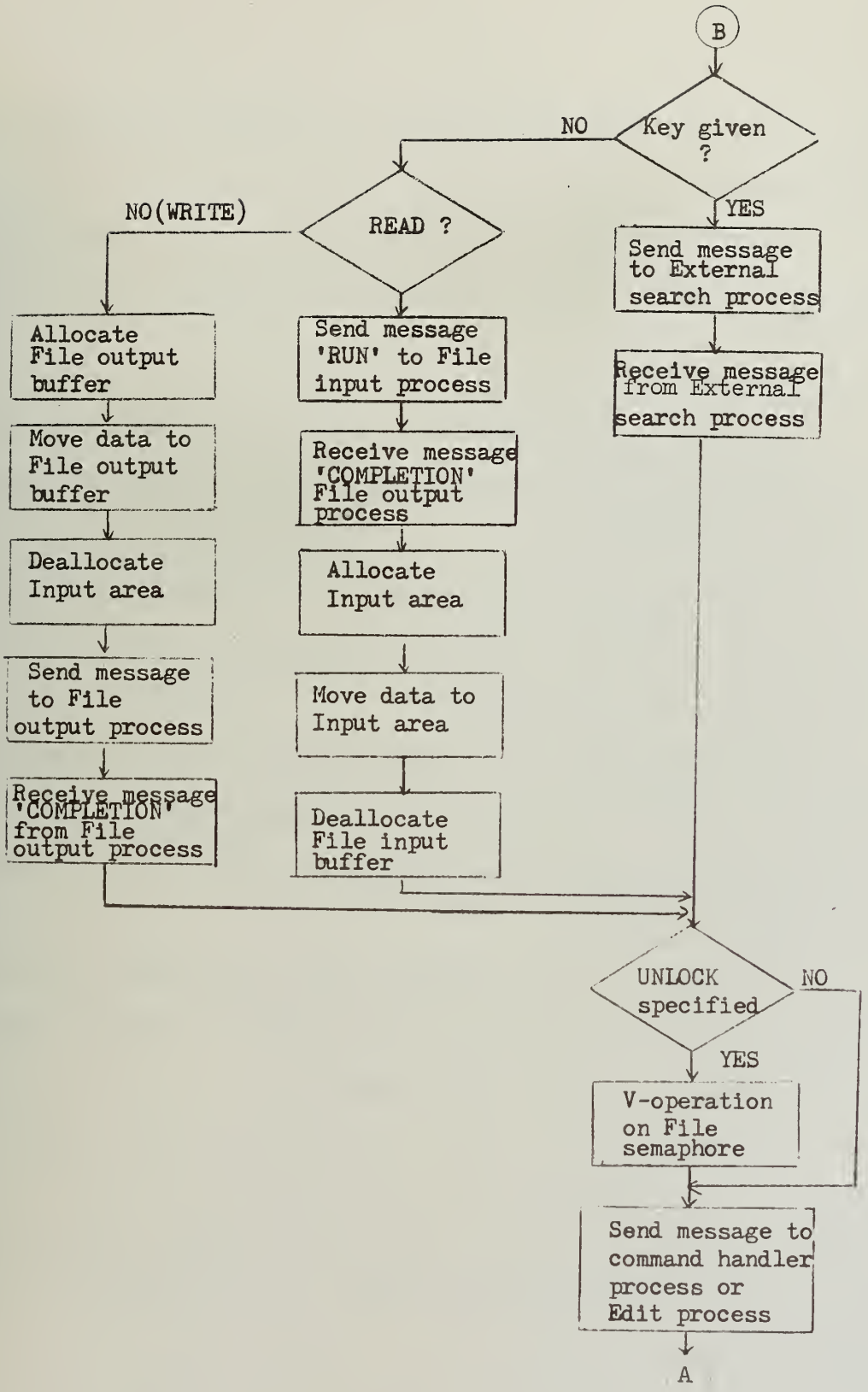


Figure 4.15 Flow of File Management Process (2)

Then if file-lock is specified, it performs a V-operation on the file semaphore.

Finally it sends a "completion" message to the command handler process and then again enters the message waiting state.

4.3.8 External Search Process

The flow of the external search process is shown in Figs. 4.16 and 4.17. This process performs the external search operation discussed in section 3.4 (see Fig. 3.6).

When it is created, it creates the probe-delete process, the file input process, and the file output process. These processes are all used in the external search operation.

This process waits for a message from the file management process. This message includes argument lists and control codes. First it computes the hash function which is in turn used to locate a bucket in the file.

After that, it sends a message to the probe-delete process and waits for an answer message from the same process. The probe-delte process performs the actual search operation.

The message from the probe-delete process is that an empty entry has been found, when the key entry is not in the file. If a write operation is required, the key insert routine is entered and then sends a message to the file output process to read-out new entry into the secondary memory.

However, if the process is informed of a key match, and if a delete operation is required, it again sends a message to the probe-delete

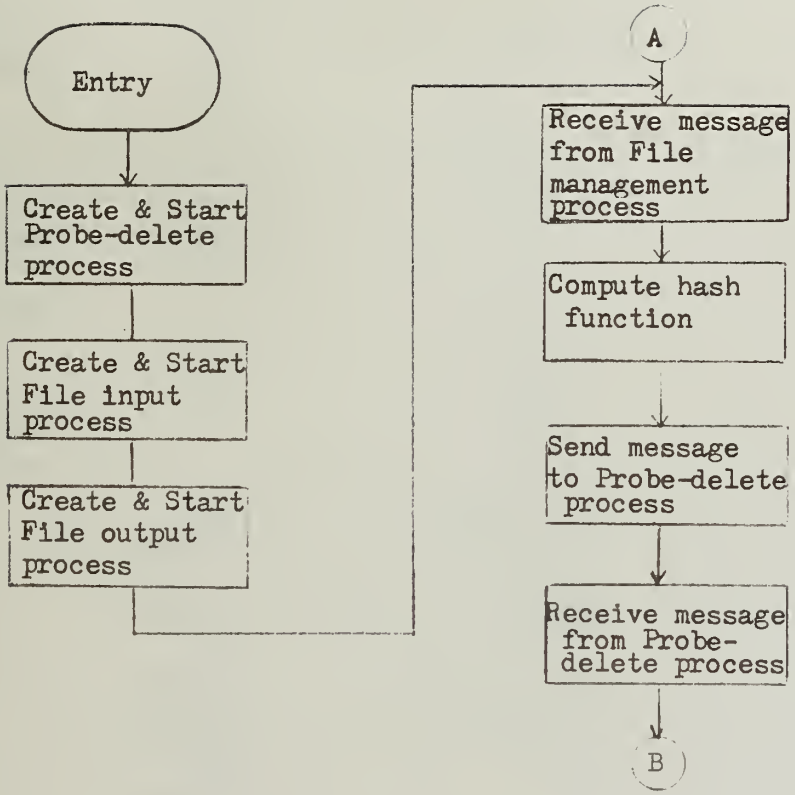


Figure 4.16 Flow of External Search Process (1)

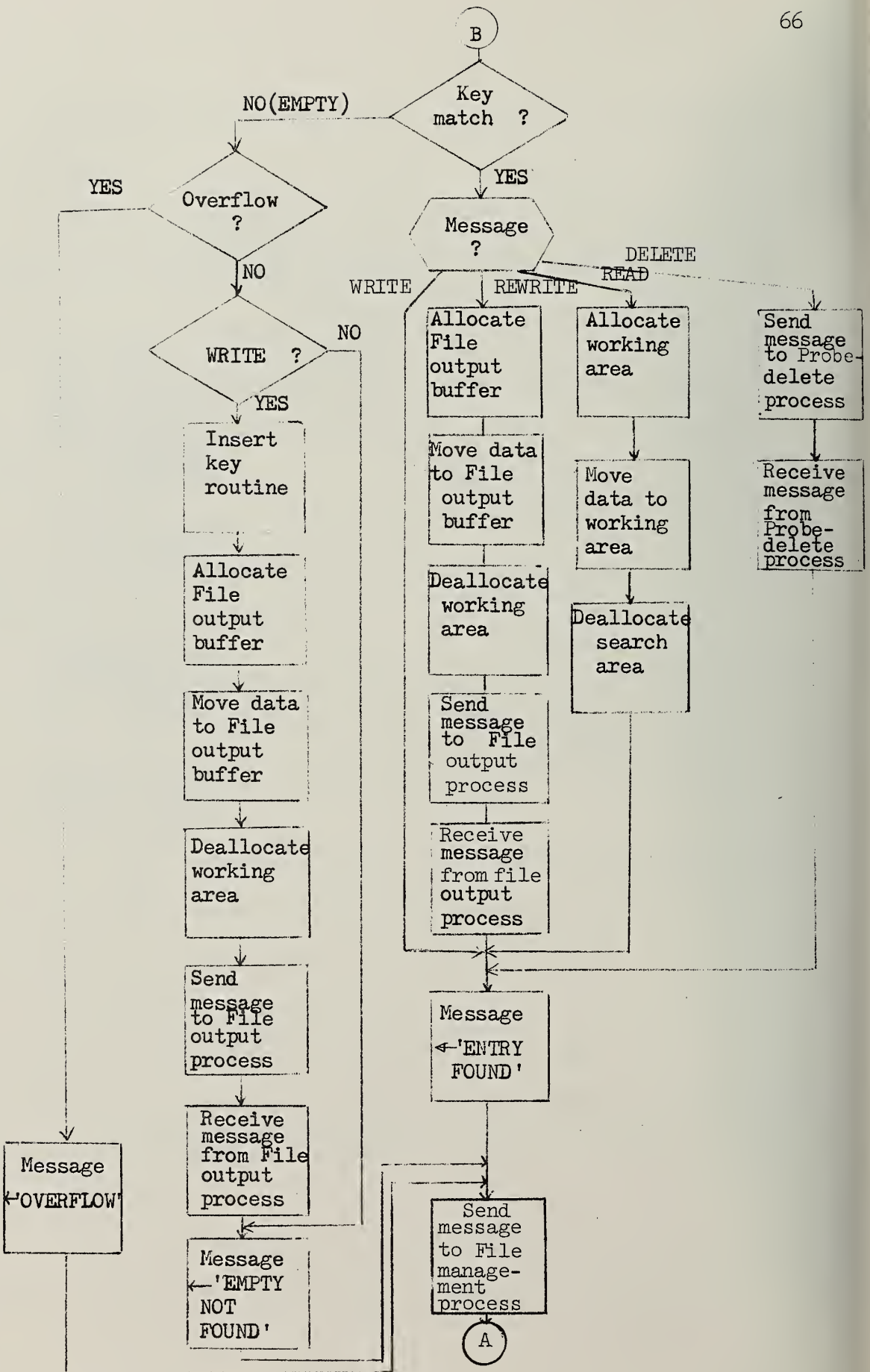


Figure 4.17 Flow of External Search Process

process to perform the deletion. If a read operation is required this process moves the data item to the working area, and a rewrite operation is required and process requires a read-out operation by sending a message to file output process.

Finally, this process sends a "completion" message to the file management process.

Although the process flow is designed based on the external search routine and the delete routine presented in section 3.4, the minor change was made to remove redundant operations. As a result, the probe-delete process is introduced.

4.3.9 Probe-delete Process

The flow of the probe-delete process is shown in Figs. 4.18 and Fig. 4.19. This process is introduced by merging probe and searching delete operations. This process performs actual search and delete operations depending on the content of message received.

After receiving a "run" message from the external search process, if the probe operation is required, this process performs the bucket read-in where the key entry is supposed to be included.

Then if a key entry or an empty entry is found, it immediately sends a message to the external search process. This message includes the resulting information.

However, if neither entry is found in the bucket, this process enters into a probing loop until it finds a key entry or an empty entry. In this case, if the delete operation is specified, the loop includes a delete routine which moves items of entries in turn until this process finds an empty entry (see Fig. 3.7).

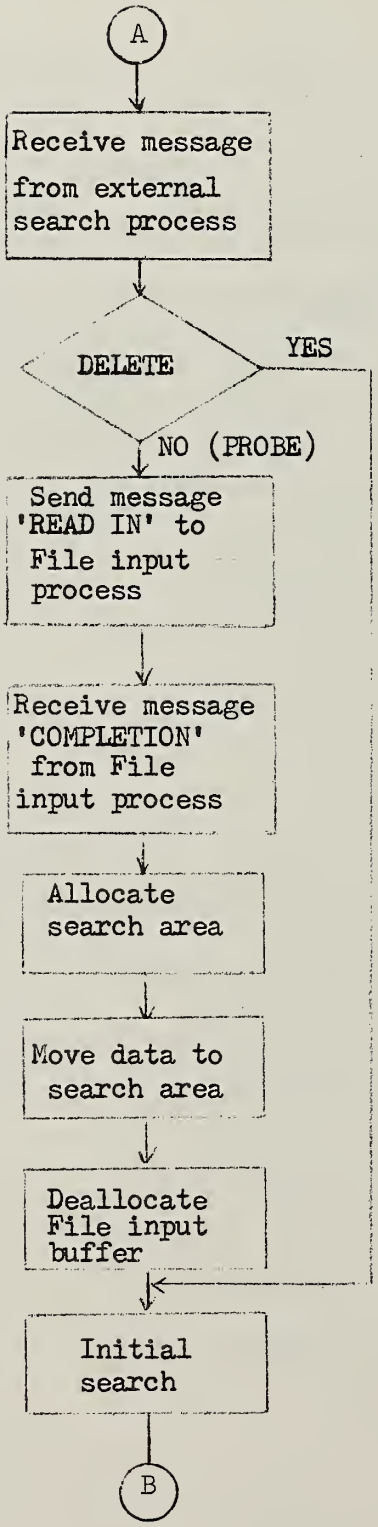


Figure 4.18 Flow of Probe-delte Process (1)

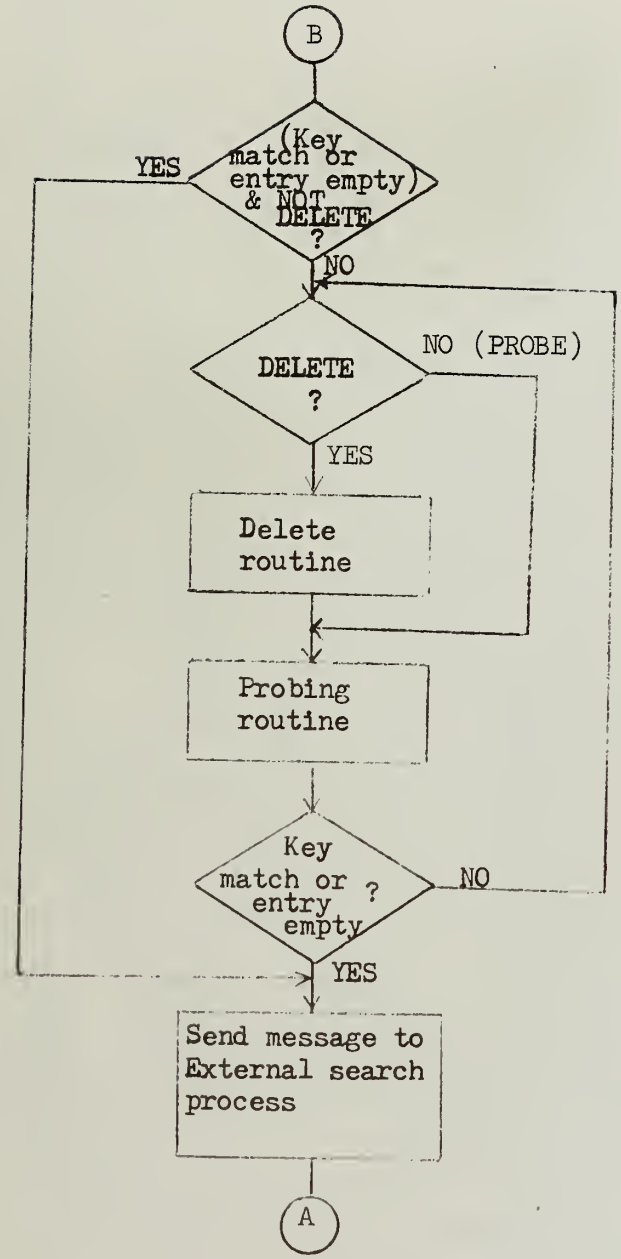


Figure 4.19 Flow of Probe-delete Process (2)

4.3.10 File I/O Process

The file input process performs a file read-in operation while the file output process performs a file read-out operation. The operation of both processes is based on a block transfer operation. Hence, the operations of blocking and deblocking must be included in these processes.

4.4 Hardware Architecture and Operating System

The approach employed in the design of the reservation system is to assign a dedicated processor or subnetwork to each of these processes described in section 4.3.

4.4.1 The Rules of Organizing Cluster Network

The rules of organizing cluster network are as follows. Each process which was decomposed from the system runs on the dedicated node processor of cluster network configuration. One method of realizing inter-process communication is via messages which are transmitted on high speed communication lines between processors. The other method for realizing inter-process communication is to use a common data-base. In this case, the data-base is considered to consist of global parameters which are accessed by more than one process. The nodes in the cluster network may be further organized as multi-processor subnetworks. Each node has a local operating system and hence, functions an autonomous system. Each node can run asynchronously with respect to other nodes.

4.4.2 Cluster Network Organization

Fig. 4.20 shows the common data base in shared main memory used for the reservation system. In this figure the processors related to each

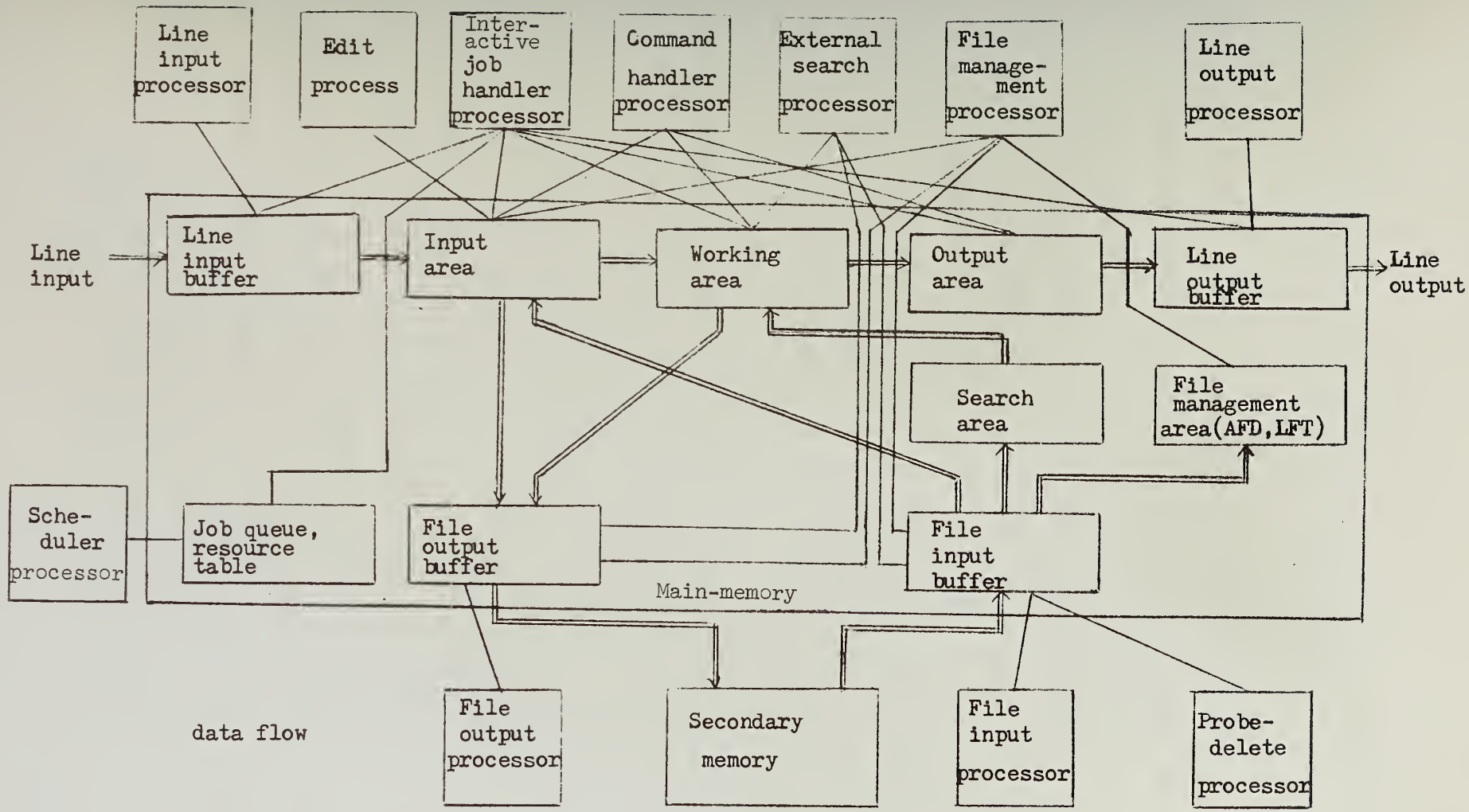


Figure 4.20 Common Database in Shared Main-memory with Related Processors

memory area are also shown. These processors allocate the corresponding memory area and work on them. The operations on these memory areas are as follows:

1. The line input data (terminal command) is read into the line input buffer by the line input processor.
2. The system job queue and the system resource table are updated by the interactive job processor and referenced by the scheduler processor.
3. The line input data in the line input buffer is moved to the input area by the interactive job handler processor.
4. If editing is required, the file data in the file input buffer is moved to the input area by the file management processor.
5. If the input data needs to be stored, the input data in the input area is moved to the file output buffer by the file management processor.
6. The input data is moved to the working area from the input area by command handler processor.
7. If an external search is required, the file data in the file input buffer is moved to the search area by the probe-delete processor.
8. The results of the search are moved to the working area from the search area by the external search processor.
9. If a rewrite operation (entry update) is required, the data in the working area is moved to the file output buffer by the external search processor.

10. The results of command handling operation are created in the output area or moved to the output area from the working area by the command handler process.
11. The output data in the output area is moved to the line output buffer by the interactive job handler processor.
12. The output data in the line output buffer is sent out to the communication line by the line output processor.

Fig. 4.21 shows the resultant cluster network configuration with shared main memory for this system. The shared main memory is connected with all processors by memory lines, whereas processors are connected to each other by inter-processor communication lines. Each processor has its own local memory where the local operating system and process routine reside. The shared main memory has memory mapping facilities and an allocation mechanism. Four processors: the line input processor, the line output processor, the file input processor, and the file output processor are considered to be I/O processors and connected to I/O devices. Note that 'port' is special hardware to connect each processor node to connection lines. All connections are implemented using dedicated lines.

Fig. 4.22 shows a cluster network configuration without shared main memory for the reservation system. In this case, the volume of message between processes is sometimes very large, because the message includes data information as well as control information. However, it offers completely homogeneous physical network configuration.

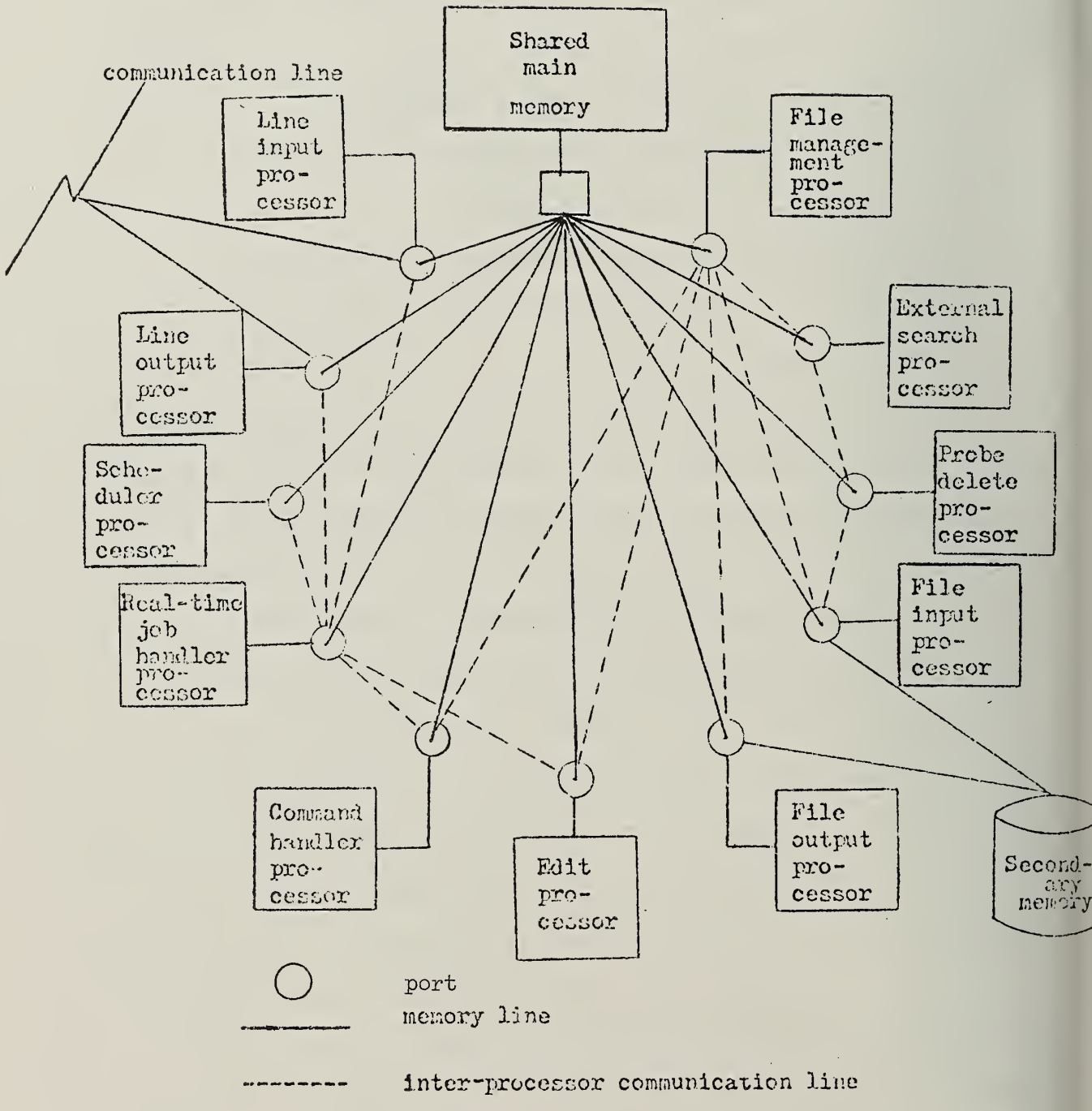


Figure 4.21 Processor Network Configuration with Shared Main Memory

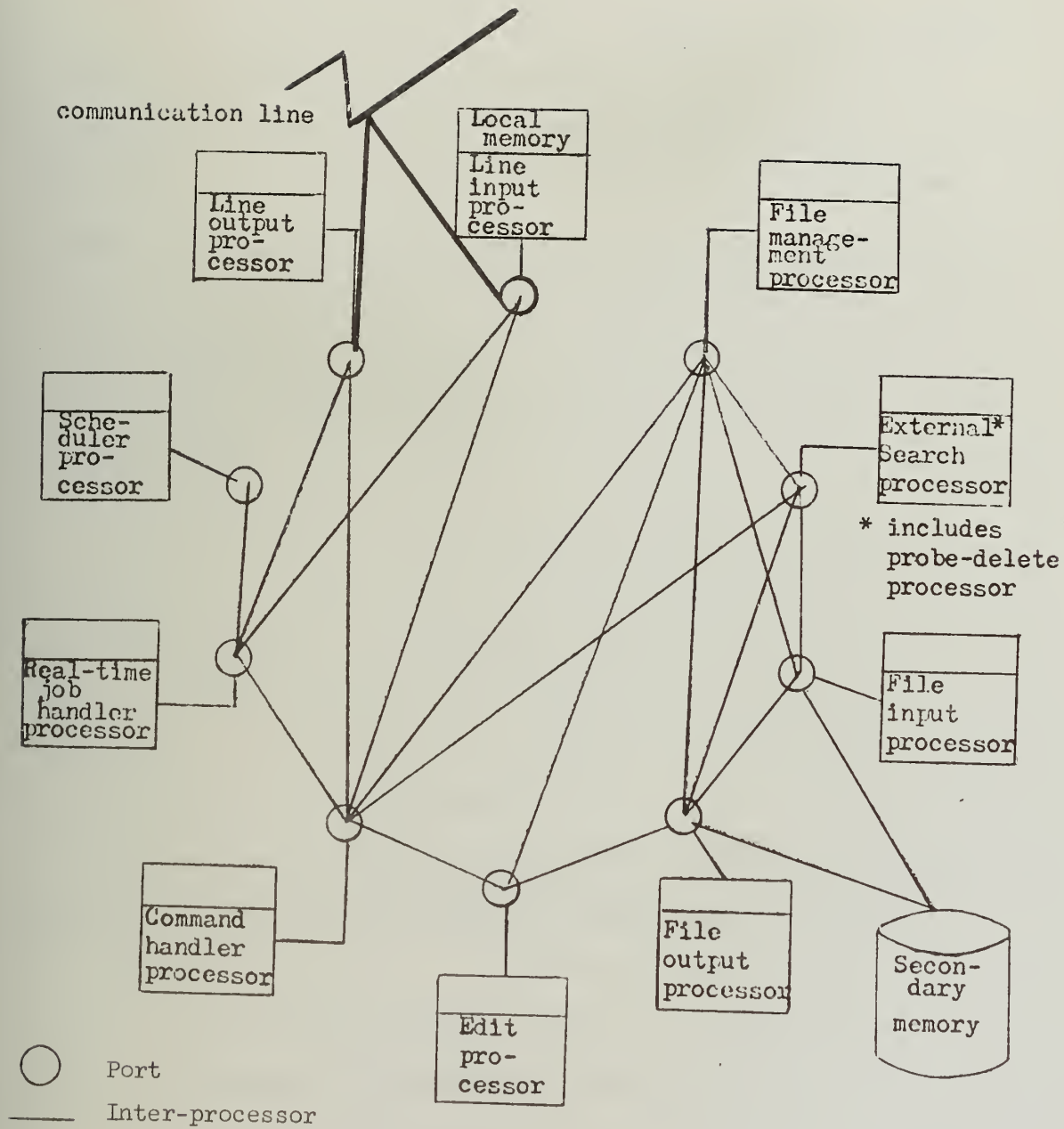


Figure 4.22 Processor Configuration Without Shared Main Memory

4.4.3 Local Operating System

Node processors run asynchronously with each other. In order to make the node processor autonomous, each node processor has its own local operating system. The local operating system includes a processor management mechanism and part of a memory management mechanism. These mechanisms are sometimes called a kernel. A kernel consists of two parts: a fixed part (program) and a variable part (data base) stored in Read Only Memory (ROM) and Random Access Memory (RAM), respectively. Also stored in ROM is, of course, the routine for the specified process.

The program routines stored in the ROM are as follows:

1. Process dispatcher (traffic controller):
   This mechanism assigns a processor to a 'READY' process in a process descriptor queue in a multi-processor environment within the node processor.
2. Supervisor call (SVC) interrupt handler:
   This mechanism handles an SVC interrupt. It includes a copy of registers and a control transfer.
3. External interrupt handler:
   This handles an external interrupt. It may change the status of a process from "BLOCKED to "RUNNING."
4. SVC routines:
   P-operation, V-operation, create process, destroy process, start process, stop process, send message, receive message, allocate memory, deallocate memory, etc.

5. The routine for assigned process:

   A process routine discussed in section 4.3 is implemented. This process routine uses SVC routines.

The data bases implemented in the RAM are as follows:

1. Process descriptor:

   A processor is represented on the data base called Process Control Block (PCB) with a unique process name. These PCBs are connected together to form a double linked list to ease insertions and deletions.

2. Message buffer:

   A message sent to a process is inserted in a queue with a header in the PCB of the receiving process. The header in the PCB includes a pointer to the first block of the queue and a message receiver semaphore.

4.4.4 Node Processor Organization

The hardware organization of node processors assumed is that of components connected on a bus. These components are as follows:

1. Micro-processor:

   The micro-processor runs on micro-codes in the ROM. The number of micro-processors attached can be increased depending on the traffic volume of the node.

2. ROM:

   Kernel routines and the routine for the assigned process reside here.

3. RAM:

   The data base on which the kernel routines are implemented is contained in the RAM.

4. PORT:

   This is a special hardware unit to connect a node processor with other node processors or I/O devices by high-speed communication lines. It consists of multiplexer gates and the controller which is also a micro-processor.

4.4.5 Implementation of Kernel Routines

The implementations of the kernel routines are almost the same as those of conventional multi-programming systems. However, additional features are still necessary with the help of hardware.

As an example, the flow of the kernel routine 'send message' is shown in Fig. 4.23. In this figure, it is assumed that some process in sending processor calls the SVC routine 'send message'. On entry to the SVC routine the sending processor sends a receiving process-name to the receiving processor with an external interrupt signal.

The receiver side is entered by the external interrupt signal, and the receiving processor tries to find the process name in the process descriptor. If there is no such process the receiver side sends a 'NO' code with external interrupt to sending processor and then returns to the calling process. However, if the process name is found, it allocates a message buffer and sends an 'OK' code with the external interrupt to sending processor.

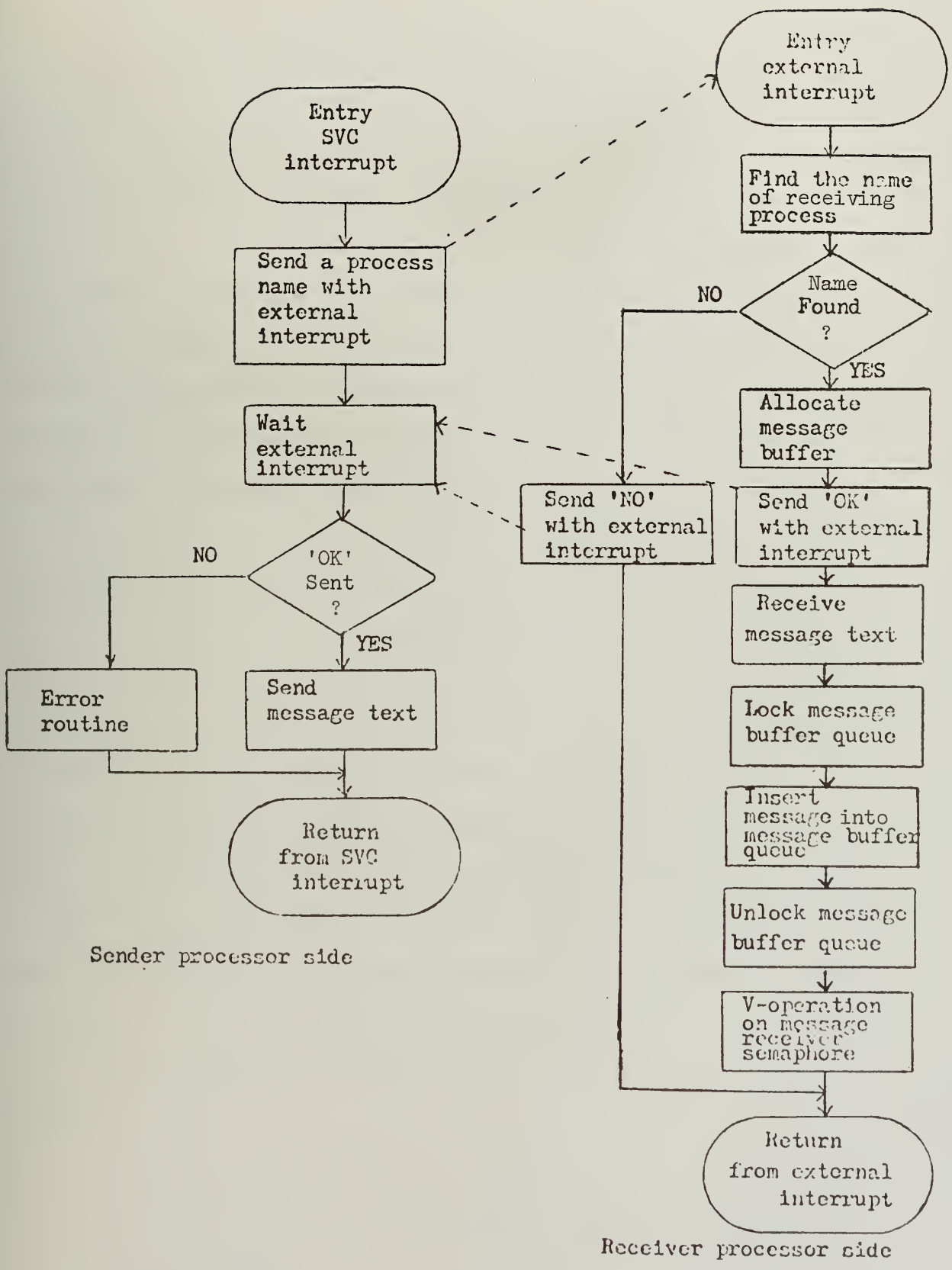


Figure 4.23 Flow of Kernel Routine 'Send Message'

The sender side which is waiting the external interrupt is awakened, and if the returned code shows that the process corresponding to the name has been found, it begins to send the message text. However, if the name has not been found the error routine is entered. Then the sender side routine returns to the calling process.

The receiver side, after receiving the message text, locks the access to the messages buffer queue and inserts the message into the message buffer queue of the receiver process. Then this routine unlocks (these lock and unlock mechanisms are implemented by P and V-operation) the access to the message buffer queue. Then it performs a V-operation on the message receiver semaphore of the receiver process. Hence, if this process is waiting, its state is changed from "BLOCKED" to "READY." After that it returns to the calling process.

4.5 Summary

In this chapter, the summary of operation for the reservation system was discussed. Then the total system design based on the process cooperation concept was made and the precise flow of each process used in this system was presented. Finally, the cluster multi-processor organization and the distributed operating system which are the final design results for this system were presented.

5. CONCLUSION

The design approach utilizing a multi-processor cluster network system based on the process cooperation concept was presented with an airline reservation system as an example. Since the actual implementation of the design was not done, the complete validity of the design cannot be guaranteed. However, the feasibility of using a multi-processor cluster network system for the application was successfully established in this paper. The design approach itself is very general and is applicable to every other system as well. For example, a time-sharing system (TSS) can be easily realized by changing the command handler process (see Fig. 4.12) according to the specification of TSS. A batch system can also be realized by adding the batch job handler process which is analogous to a batch monitor in a conventional batch system. In both cases additional processes such as a user process and a compiler process are required and corresponding processor-nodes must be added to the multi-processor cluster network. Then the user process would always be run in a general-purpose processor-node.

Future research in this area is still required for the efficient and elegant system.

This future research must include the following:

1. Establishment of a complete and consistent method to decompose the system into processes:

In this paper the process decomposition was partly done in an ad hoc manner even though the rules for the decomposition were specified. Further development of structured programming seems to be in the direction of a solution to this problem. This is a problem inherent to every kind of system design.

2. Establishment of inter-connection network design method:

   In this design, processors, memory, and I/O devices were connected in one-to-one fashion as needed. However, the amount of inter-connection hardware sometimes becomes a bottleneck of hardware cost. The topology of minimum hardware network including a use of buses must be sought. Research work in this area seems to be advancing [4].

3. Development of micro-processor architecture:

   Present micro-processors can be used as node-processors. However, it is desirable to develop further a micro-processor which has an ability to send and receive a vast amount of inter-process messages, i.e., a micro-processor with multiple ports.

LIST OF REFERENCES

[1] Davis, R. L., S. Zucker and C. M. Campbell, "A Building Block Approach to Multiprocessing," Proceedings, AFIPS, 1972, SJCC, Vol. 40, pp. 685-703.

[2] Baskin, H. B., B. P. Borgerson and R. Roberts, "PRIME-A Modular Architecture for Terminal-Oriented Systems," Proceedings, AFIPS, 1972, SJCC, Vol. 40, pp. 431-437.

[3] Gillies, D. B., et al., "MESH Project First Annual Report," Quarterly Technical Progress Report, UIUCDCS-OPR-73-4, Department of Computer Science, University of Illinois, Nov., 1973, pp. 102-127.

[4] Chesson, G. L., "Communication and Control in a Cluster Network," Unofficial report, Department of Computer Science, University of Illinois, 1974.

[5] Madnick, S. E. and J. W Alsop, "A Modular Approach to File System Design," Proceedings, AFIPS, 1969, SJCC, Vol. 34, pp. 1-14.

[6] Knuth, D. E., The Art of Computer Programming: Sorting and Searching, Vol. 3, Addison-Wesley, Reading, Mass., 1973, pp. 504-569.

[7] Hansen, P. B., "The Nucleus of a Multiprogramming System," Comm. ACM, Vol. 13, No. 4, April, 1970, pp. 238-250.

[8] _____, Operating System Principles, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973, p. 385.

[9] Dijkstra, E. W., "The Structure of the "THE"-Multiprogramming System," Comm. ACM, Vol. 11, No. 5, May, 1968, pp. 341-346.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-75-717	2.	3. Recipient's Accession No.

4. Title and Subtitle

Multiprocessor Network System Design by Process Concept

5. Report Date: December, 1975

6.

7. Author(s)

Akihiro Iwaya

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois
Urbana, Illinois

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Address

Department of Computer Science
University of Illinois
Urbana, Illinois

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

This report describes the design of a dedicated multiprocessor network which functions as an airline reservation system. The design approach used is as follows: The system is decomposed into cooperative processes. A dedicated processor is assigned to each process and processors are connected via suitable inter-process communication links to form a cluster network. Many aspects in operating system design, such as protection mechanisms, file management, etc. are discussed as well.

17. Key Words and Document Analysis. 17a. Descriptors

multiprocessors
information systems
interprocessor communications

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement

19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
20. Security Class (This Page UNCLASSIFIED	22. Price

BOUND BY
HNM

UNIVERSITY OF ILLINOIS-URBANA
3 0112 047424251